SINDH TEXTBOOK BOARD, JAMSHORO

THE TEXTBOOK OF

COMPUTER SCIENCE

For Grade 10

SINDH TEXTBOOK BOARD, JAMSHORO

Prepared by **ASSOCIATION FOR ACADEMIC QUALITY (AFAQ)** for SINDH TEXT BOOK BOARD JAMSHORO.

Reviewed by Provincial Review Committee Directorate of Curriculum Assessment and Research Sindh Jamshoro (DCAR).

Prescribed as a Textbook by the **Boards of Intermediate and secondary Education, Karachi, Hyderbad, Sukkur, Larkana Mirpurkhas and Shaheed Benazirabad** for Secondary School Certificate Examination in the Province of Sindh.

Approved by the **Education and Literacy Department, Government of Sindh.**
No. SO(C)SELD/STBB-18/2021 Dated: 14th July, 2021 for the province of Sindh.

Patron in Chief
Pervaiz Ahmed Baloch
Chairman, Sindh Textbook Board.

Shahid Warsi
Managing Director
Association For Academic Quality (AFAQ)

Khawaja Asif Mushtaq
Project Director
Association For Academic Quality (AFAQ)

Rafi Mustafa
Project Manager
Association For Academic Quality (AFAQ)

Yousuf Ahmed Shaikh
Cheif Supervisor
Sindh Textbook Board.

Supervisor
Daryush Kafi
Sindh Textbook Board, Jamshoro

AUTHORS

- **MS. Zufishan Kamal**
- **Mr. Ajmal Saeed**
- **Mr. Hanif Ahsan Zubedi**

REVIEWERS

- **Mr. Abdul Majeed Bhurt**
- **Professor (Retd.) Muhammad Zahid Shaikh**
- **Mr. Imran Pathan**
- **Mr. Amjad Ali Yousuf Zai**
- **Mr. Mushtaque Ahmed Ansari**

Editor

- **Khawaja Asif Mushtaq**

TECHNICAL ASSISTANCE CO-ORDINATOR

- **Mr. M. Arslan Shafaat Gaddi**

DESIGNING & ILLUSTRATION

- **M. Arslan Chauhan**

Printed at:

PREFACE

After textbook of Computer Science for Grade 9, Computer Science for Grade 10 is now ready to be used by teachers and students. This textbook has been developed on Curriculum of Computer Science 2018 reviewed by Directorate of Curriculum, Assessment and Research Sindh, Jamshoro.

We expect that this book will fulfill the diverse needs of students studying public and private schools across Sindh. This textbook encompasses some very important of skill required in 21st century like Algorithm Designing, Problem Solving, Logic Design and above all Coding. To compete with the rapidly changing world we need to equip our youth with these skills.

IT is an area from where we have great potential. With skillful youth we can generate a lot of foreign exchange even if we have very limited resources. Teachers can play and an important role to equip students with IT skills and this textbook will be helpful for them.

We believe that this book will help student to imagine and perceive beyond this textbook, and instead of cramming the knowledge given in the book, they will make efforts to develop and strengthen their own ideas and skills.

Our organization is indebted to Association For Academic Quality (AFAQ), all the Authors and Reviewers of this book who made rigorous efforts to deliver a book that is competitive with any other textbook at this level. We encourage teachers, students, parents, researchers and all the stakeholders to give their feedback and suggestion to further improve this book.

Chairman
Sindh Textbook Board, Jamshoro

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

CONTENTS

PROBLEM SOLVING AND ALGORITHM DESIGNING

- Define the term problem
- Evaluate a problem in order to find out its best solution
- Design a strategy for the solution of problem
- Find feasible solutions of a problem

1.1 PROBLEM SOLVING

Problem solving is the process of finding solutions of difficult or complex issues. It is the process by which any kind of problem is solved.

Solving problems is the core feature of computers. A computer is not intelligent. It cannot analyze a problem and come up with a solution. A human (the programmer) must analyze the problem, develop the instructions for solving the problem, and then make the computer carry out the instructions. The major responsibility of a programmer is to provide solution of the problems by using computers. It will be easier if computer science students first understand how a human solves a problem, then understand how to translate this solution into something a computer can understand, and finally how to "write" the specific steps to get the job done. Remember, in some cases a machine will solve a problem in a completely different way than a human.

What is Problem

A problem is a situation preventing something from being achieved. A problem can be a task, a situation or any other thing. In simple language, a problem is a question which requires an answer or a solution. In any case, a problem is considered to be a matter which is difficult to solve or settle, a doubtful case, or a complex task involving doubt and uncertainty.

Plan the solution of Problem

Problems can be solved with the help of computers but for this programmer has to plan and strategize tasks that lead to the solution of the problem.

Problem Solving Strategies

A very important aspect of problem-solving is developing good strategies. There are many strategies for solving a problem. A strategy is an approach (or a series of approaches) created to solve a computational problem. Strategies are designed according to the nature of the problem. Strategies are flexible and show various steps to reach the solution. A strategy in itself might produce incorrect results, but an algorithm based on such strategy will always produce correct results.

For solving any problem, it is important to know what the problem really is and what how it should actually be if there was no such problem. The question about end result helps you to find the gap and create a strategy for solution.

Problem Solving Process

Problem solving is a step-by-step process. There are four basic step involved in finding a solution for a problem as given below:

1. Define the problem.
2. Generate alternative solutions.
3. Evaluate and select an alternative.
4. Implement and follow up on the solution.

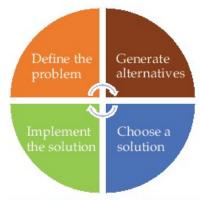



Fig. 1.1: Problem Solving

1. Define the problem

In problem solving process, the first step is defining or identifying the problem. It is the most difficult and the most important of all the steps. It involves diagnosing the situation so that the focus should be on the real problem and not on its symptoms. During this first stage of problem solving, it is important to describe the problem. A well- described problem will also help others to understand the problem.

2. Generate alternative solutions

For any problem, there are more solutions to it than the one that is thought of first. Postpone the selection of one solution until several problem-solving alternatives have been proposed. Considering multiple alternatives can significantly enhance the value of your ideal solution. So, it is best to develop a list of all feasible solutions that can be assessed and decide which one will be the best for the particular problem. Thinking and team problem-solving techniques are both useful tools at this stage of problem solving.

3. Evaluate and select an alternative

Many alternative solutions to the problem should be generated before final evaluation. A common mistake in problem solving is that alternatives are evaluated as they are proposed, so the first acceptable solution is chosen, even if it is not the best solution. If we just focus on trying to get the results we want, we can miss the chances for learning something new which is required for real improvement in the problem-solving process.

4. Implement and follow up on the solution

The plan for best solution also includes planning on what happens next if something goes wrong with the solution if it does not work out the way it was required. When the best solution is implemented, it is important to track and measure the results to be able to answer questions such as: Did it work? Was this a good solution? Did we learn something here in the implementation that we could apply to other potential problems?

When choosing most appropriate solution, the problem solver should consider about the possible impacts of that solution. For example, will this solution solve the current problem without creating new ones or if this solution is acceptable by everyone involved in this situation or if the solution is within the budget and achievable within a given time.

- Define term algorithm
- Discuss the importance of algorithm in problem solving

1.2 ALGORITHM

Algorithms are widely used throughout all areas of Information Technology (IT). A search engine algorithm, for example, takes search strings of keywords and operators as input, searches its associated database for relevant web pages, and returns results. Another common example is online advertisements where the algorithm takes age, gender, region and interests as input and then displays ads to only those people who match the criteria.

1.2.1 Definition

An algorithm is a set of instructions/steps/rules that are followed to solve a problem. It is a tool for solving a well-specified computational problem. In our daily lives as well, we follow various algorithms without knowing, for example:

- We follow a daily routine after waking up in the morning
- We follow a set of instructions while driving a car
- We follow recipes in cooking food or making tea

These step-by-step instructions that we follow everyday are algorithms to solve certain problems. There are two common methods to express algorithm designs; pseudocode and flowcharts.

1.2.2 Role of algorithm in problem solving

The advantage of using an algorithm to solve a problem or make a decision is that it produces the best possible answer every time. This is useful in solutions where accuracy is required or where similar problems need to be solved more often. In many cases, computer programs can be developed with the help of this process. Then data is entered in that program so that the algorithm can be executed to come up with the required solution.

A computer programmer has to design various algorithms to create a program. These algorithms can vary from retrieving input from a user to

computing complex formulas to reach a conclusion. This data is then rearranged into a meaningful way to present to the user. The user then makes decisions based on the presented data.

Qualities of Good Algorithms

- Input and output should be defined precisely.
- Each step in the algorithm should be clear and unambiguous.
- Algorithms are supposed to be most effective among many different ways to solve a problem.
- An algorithm should not include computer code. Instead, the algorithm should be written in such a way that it can be used in different programming languages.

♦ Design algorithm to find sum, average, volume, percentage and others.

1.2.3 Algorithm Examples

Following are few examples of simple algorithm.

Algorithm 1: Making a cup of tea

- **Step 1:** Start
- **Step 2:** Place the fresh water in a pot or a kettle.
- **Step 3:** Boil the water.
- **Step 4:** Put the black tea leaves in that pot.
- **Step 5:** After that add some milk into that pot.
- **Step 6:** Add some sugar.
- **Step 7:** Boil for some time.
- **Step 8:** Stop

Algorithm 2: Sum of two numbers

- **Step 1:** Start
- **Step 2:** Declare variables num1, num2 and sum.
- **Step 3:** Read values num1 and num2.
- **Step 4:** Add num1 and num2 and assign the result to sum.
- sum = num1 + num2
- **Step 5:** Display sum
- **Step 6:** Stop

Algorithm 3: Average of three numbers

- **Step 1:** Start
- **Step 2:** Declare variables num1, num2, num3 and avg.
- **Step 3:** Read values num1, num2 and num3.
- **Step 4:** Apply formula {Average = Sum / No. of values}
- avg = (num1 + num2 + num3) / 3
- **Step 5:** Display avg
- **Step 6:** Stop

Algorithm 4: Volume of a box

- **Step 1:** Start
- **Step 2:** Declare variables length, width, height and volume.
- **Step 3:** Read values length, width and height.
- **Step 4:** Apply formula {Volume = length x width x height}
- volume = length x width x height
- **Step 5:** Display volume
- **Step 6:** Stop

Algorithm 5: Percent Calculate

- **Step 1:** Start
- **Step 2:** Declare variables part, total and percentage.
- **Step 3:** Read values part and total.
- **Step 4:** Apply formula {Percentage = (part / total) × 100}
  percentage = (part / total) x 100
- **Step 5:** Display percentage
- **Step 6:** Stop

- Define the flowchart
- Identify the different symbols used in flowchart designing
- Discuss the importance of flowchart in problem solving
- Design flowchart for any problem by using various flowchart symbols
- Differentiate between algorithm and flowchart

1.3 FLOWCHART

The flowchart is the physical representation of problem solving process. It is used to show the sequence of steps and logic of solution for a problem, before writing the full computer program. It also helps in communicating the steps of the solution to others by using different symbols.

1.3.1 Definition

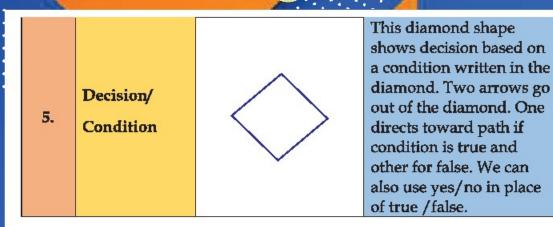
It is a general-purpose tool used to define the sequence of different types of processes or operations in information system or program. It shows processes and their flow visually using diagram. It describes graphically different steps of programs or any operation and their sequence or flow using different symbols. Information system flowcharts show flow of data from source documents to final distribution to users. Program flowcharts show the sequence of steps or instructions in a single program or subroutine. It is diagrammatic or graphical representation of algorithm and converts word of algorithm into symbols. The flowchart shows different steps with the help of

different shapes and their sequence. The processes are connected with directed lines or arrows which show the path from one procedure step to the next.

1.3.2 Flowchart Symbols

Flowchart is made up of different symbols to represent or show program and its flow. Some of them are as follows:

S No.	Name	Symbol	Description
1.	Start/Stop	Start Stop	It is oval shape and is used to show the start and end of a program or flowchart sequence.
2.	Arrows		The arrow shape shows direction of flow from one step or box to another.
3.	Process		This rectangle shape indicates any type of internal operation or process usually one step. The step is written inside the box. Only one arrow goes out of the box.
4.	Input/output		This parallelogram shape shows the input or output process. It is used for any Input / Output (I/O) operation and indicates that the computer is to obtain data or output results.

5.	Decision/ Condition		This diamond shape shows decision based on a condition written in the diamond. Two arrows go out of the diamond. One directs toward path if condition is true and other for false. We can also use yes/no in place of true /false.

Teachers Note	
	Teacher should tell students about other symbols like alternate symbol for start/end, connecter symbol, etc.

1.3.3 Importance of a Flowchart for Solving a Problem

Main use of a flowchart is to visualize operations and sequence of steps to perform them. It illustrates the logic to solve a problem, before writing the full computer program. It shows all steps visually or graphically which is easy to understand in one look and also helps to describe program flow to others. It also helps programmers when we modify or extend program. Following are some advantages of flowchart for solving problem

- Flowcharts help in communicating the logic of a program to all others and make it easy to understand.
- It is a useful program document that is needed for various purposes like to know about program quickly or to modify program logic.
- The flowcharts act as a guide or blueprint during the coding of program.

A Sample Flowchart

It takes three numbers as input, calculates sum and percentage. If percentage is above 70 then prints "Well done", otherwise "Work hard".

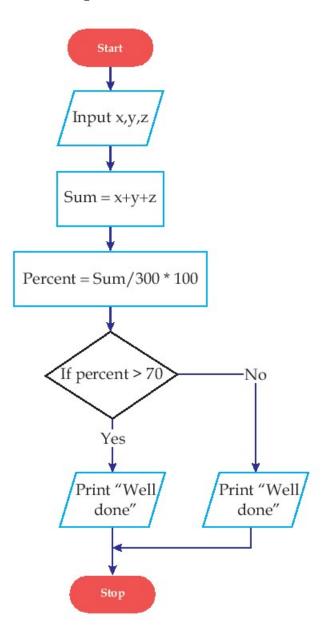



Fig. 1.2: Sample Flowchart

1.3.4 Difference between Algorithm and Flowchart

S.No.	Algorithm	Flowchart
1	Algorithm is step by step solution of a problem	Flowchart is a diagram of different shapes which shows flow of data through processing system.
2	In algorithm text is used.	In flowchart, symbols or shapes are used
3	Algorithm is easy to debug.	Flowchart is difficult to debug
4	Algorithm is difficult to write and understand.	Flowchart is easy to construct and understand.
5	Algorithm does not follow any rules.	Flowchart follows rules for its construction.
6	Algorithm is the pseudo code of the program.	Flowchart is just graphical or visual representation of program logic.

- Define Linear data types
    - Stack, Queue, Array
- Define Non-Linear data types
    - Tree & Graph

1.4 DATA STRUCTURE

A **data structure** is a particular way of organizing data in a computer to use it effectively. For example, array data structure is used to store a list of items having the same data-type. Data structure may be linear or non-linear.

1.4.1 Linear Data structures

In Linear Data Structure data elements are arranged in sequential order and each of the elements is connected to its previous and next element. This structure helps to convert a linear data structure in a single level and in single run. They are easy to implement as computer memory is also in a sequential form. Linear data structures are not efficient in memory utilization. Examples of linear data structures are Stack, Queue, Array etc.

(a) Stack

Stack is a linear data structure which follows a particular order to perform different operations. Items may be added or removed only at the top of stack. The order may be LIFO (Last In First Out) or FILO (First In Last Out). The data which is placed first is removed in last and which is placed last is removed first. We cannot remove data from the bottom or middle. Examples of a stack are plates that are stacked or put over one another in the canteen. The plate which is at the top most is the first one to be removed and the plate which is placed at the bottom most position remains in the stack until last plate is removed.

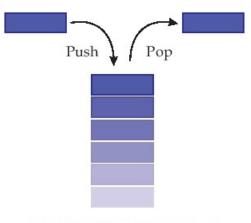



Fig. 1.3: Stack (Push- Pop)

The term *push* is used to insert a new element into the stack and *pop* is used to remove an element from the stack. Insertion and removal can be done at one end called top. Stack is in overflow state when it is completely full and is in underflow state if it is completely empty. In overflow state we cannot add an item and in underflow state we cannot remove an item from it.

(b) Queue

A Queue is a linear data structure which follows a particular order in which operations are performed in FIFO (First In First Out) method, which means that element inserted first will be removed first. Example of a queue is any queue of students in school or people in cinema where one who comes first gets the ticket first. Deletions take place at one end called *front or head* and insertions take place only at the other end called *rear or tail*. Once a

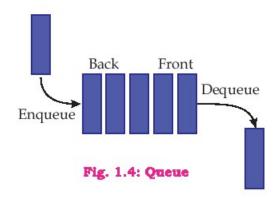



Fig. 1.4: Queue

new element is inserted into the Queue, it cannot be removed until all the elements inserted before it in the queue are removed. The process to add an element into queue is called Enqueue and the process to remove an element from queue is called Dequeue.

(c) Array

Array is a **linear data structure,** which holds a list of finite data elements of same data type. Each element of array is referenced by a set of index of consecutive numbers. The elements of array are stored in successive memory locations. Most of the data structures make use of arrays to implement their algorithms. Two terms are necessary to understand array.

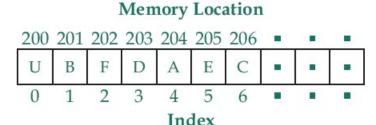



Fig. 1.5: Array (Memory Locations)

- **Element:** Each item stored in an array is called an element.
- **Index:** Each location of an element in an array has a numerical value called index, which is used to identify the element. Set of indexes in an array are consecutive numbers.

Operations like Traversal, Search, Insertion, Deletion and Sorting can be performed on arrays.

Teachers Note	Teacher should tell students about operations performed on an array like traversal, sorting, etc.

1.4.2 Non-Linear Data Structures

The elements of a non-linear data structure are not connected in a sequence. Each element can have multiple paths to connect to other elements. They support multi-level storage and often cannot be traversed in single run. Such data structures are difficult to implement but are more efficient in utilizing computer memory. Examples of non-linear data structures are Tree, Graphs etc.

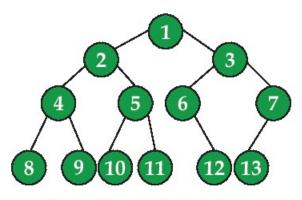



Fig. 1.6: Tree Data Structure

(a) Tree

This non-linear data structure is used to represent data containing a hierarchical relationship between elements. Tree represents its elements as the nodes connected to each other by edges. In each tree collection, we have one root node, which is the very first node in our tree. If a node is connected to another node element, it is called a parent node and the connected node is called its child node. There is also a binary tree or binary search tree. A binary tree is a special data structure used to store data in which each node can have a maximum of two children. Each node element may or may not have child nodes.

(b) Graph

A graph is a **non-linear data structure** consisting of data elements (finite set) called nodes/vertices and edges that are lines that connect any two nodes in that *graph*. Each element or node can contain information like roll number, name of student, marks, etc. In graph each node can have any number of edges, there is no any node

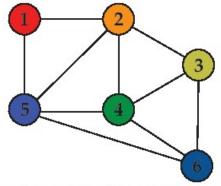



Fig. 1.7: Graph Data Structure

called root or child. A cycle can also be formed. In the given figure, circles represent nodes or vertices, while lines represent edges.

Graphs are used to solve network problems. Examples of networks include telephone networks, social networks like Facebook, etc. For example, a single mobile phone is represented as a node (vertex) whereas its connection with other phones can be shown as an edge between nodes.

Types of graphs:

There are two types of graphs.

1. **Undirected Graph:**

In an undirected graph, nodes are connected by edges that are all bidirectional. For example if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.

2. **Directed Graph:**

In a directed graph, nodes are connected by directed edges – they only go in one direction. For example, if an edge connects node 1 and 2, but the arrow head points towards 2, we can only traverse from node 1 to node 2 – not in the opposite direction.

- Problem solving is the process of finding solutions of difficult or complex issues.
- A problem is a situation preventing something from being achieved.
- There are four basic steps involved in finding a solution; define the problem, generate alternative solutions, evaluate and select an alternative and implement and follow up on the solution.
- It is important to understand the problem and set a starting point of solution.
- There are various strategies that can be used to formulate an algorithm for solving the problem.

- Using the strategy, various solutions to a given problem are planned and the most feasible solution is identified.
- Algorithm is a technical term for a set of instructions for solving a problem or sub-problem.
- Algorithms enable breaking down of problems and conceptualize solutions step-by-step.
- Algorithms are defined as generic steps of instructions so they can be written in any programming language.
- A flowchart writes the sequence of steps and logic of solving a problem, using graphical symbols.
- Flowcharts help in communicating the logic of a program to all others and make it easy for understanding.
- A **data structure** is a particular way of organizing data in a computer to use it effectively.
- In *Linear data structure* data elements are arranged in sequential order and each of the elements is connected to its previous and next element.
- Stack is a linear data structure in which items may be added or removed only at one end i.e. at the top of stack.
- Queue is a linear data structure in which that element inserted first will be removed first.
- Array is a **linear data structure,** which holds a list of finite data elements of same data type. Each element of array is referenced by a set of index of consecutive numbers.
- The elements of a non-linear data structure are not connected in a sequence. Each element can have multiple paths to connect to other elements.
- Tree non-linear data structure is used to represent data containing a hierarchical relationship between elements.
- A graph is a **non-linear data structure** consisting (finite set) of data elements called nodes/vertices and edges that are lines that connect any two nodes in that *graph.*

A. ENCIRCLE THE CORRECT ANSWER:

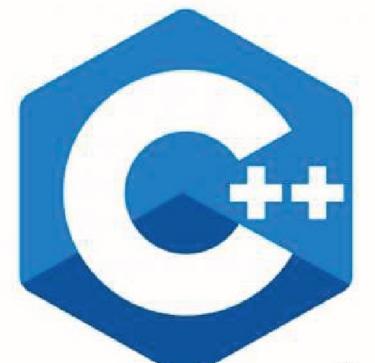
1. To "bake a cake" is an example of:
   - a. problem b. strategy
   - c. algorithm d. solution
2. To find a feasible solution to a problem, the first step is to:
   - a. establish starting point b. find available solutions
   - c. create a strategy d. identify and analyze the problem
3. Step by step solution of a problem in simple language is called:
   - a. Problem Solving b. Algorithm
   - c. Flowchart d. Data Structure
4. __________ shows the logic of program graphically.
   - a. Data Structure b. Graph
   - c. Algorithm d. Flowchart
5. ______ Symbol is used for input/output in flowchart.
   - a. Triangle b. Square
   - c. Parallelogram d. Rectangle
6. Elements of _________ data structure are not connected sequentially.
   - a. Array b. Graph
   - c. Queue d. Stack
7. ___________ stores data in hierarchal manner.
   - a. Stack b. Queue
   - c. Array d. Tree
8. When that data is Pushed in stack, it means that data is:
   - a. inserted b. deleted
   - c. sorted d. edited
9. In binary tree, each child can have maximum:
   - a. one node b. two nodes
   - c. three nodes d. four nodes
10. Traversing an array means accessing:
    - a. first element b. last element
    - c. any specific element d. each and every element of the array

B. RESPOND THE FOLLOWING:

1. Describe the steps involved in problem solving.
2. What are the advantages of developing algorithms?
3. List any three advantages of designing flowcharts.
4. What is the difference between tree and graph data structure?
5. What is the difference between queue and stack data structure?
6. What is the need of index in an array?
7. With the help of a sketch define: Push, Pop, Overflow and Enqueue, Dequeue.

LAB ACTIVITIES

1. Design an algorithm to find the greater number by taking two numbers as input.
2. Design an algorithm to find area of a triangle.
3. Sort the following steps of the algorithm in correct order for baking a cake:
    - Step: Gather the ingredients
    - Step: End
    - Step: Grease a pan
    - Step: Preheat the oven
    - Step: Put the pan in the oven
    - Step: Start
    - Step: Pour the batter into the pan
    - Step: When the timer goes off, take the pan out of the oven
    - Step: Set a timer
    - Step: Mix together the ingredients to make the batter
4. Draw a flow chart to calculate gross salary by adding 20% house rent and 30% medical allowances in basic salary.
5. Draw a flow charts for all the algorithms given in this unit.
6. Draw the following structure
    a. Tree with six nodes.
    b. Graph with five nodes.
7. Search about **Algorithmic Thinking** and in groups, discuss this concept.

Unit 2

- Define computer program
- Describe the importance of syntax in any programming language

2.1 INTRODUCTION

A computer program is a set of instructions that is understood by a computer to perform tasks. A person who writes computer programs is known as a programmer. Computer processes instruction in binary language. Therefore, programs are written in programming languages. Programming languages have a specific set of words called syntax to create those instructions. Specialized programs such as compiler are used to convert set of syntaxes into set of machine-readable instructions. It requires an interface to convert commands from a human user. A programmer can make mistakes in syntax while writing a program or instructions. Other specialized programs, known as Integrated Development Environments (IDEs), help programmers write programs in various languages. C++ is one of the most common and powerful general purpose programming language. All programming languages have certain concepts about rules of syntaxes, reserved words and data types.

2.1.1 Computer Program

A computer program is a collection of instructions that can be executed by a computer to perform a specific task. It is very difficult to write in the ones and zeroes of machine code, which is what the computer can understand, so computer programmers use specialized languages to communicate with computers to perform a set of specific tasks using languages like C++, Java or Python. Once it is written, the programmer uses a compiler to translate it into a language that the computer can understand.

Syntax in Programming Language:

Syntax tells the computer how to read a set of code. It is essentially a set of keywords and characters that a computer can read, interpret, and convert into tasks needed. Text-based computer languages are based on sequences of characters, while visual programming languages like Visual Basic are based on the layout and connections between symbols (which may be textual or graphical).

Example:

```
cout << "Hello World";
```

In C++, this syntax displays the message "Hello World" on the screen. Syntax plays an important role in the execution of programs in text-based programming languages and can even cause syntax errors if a programmer tries to run a program without using proper syntax. It is very common for new programmers to make syntax-based mistakes. Different programming languages use different types of syntax.

- Classify different programming languages into High, middle and low-level languages on the basis of their characteristics

2.1.2 Classification of Programming Languages

Thousands of programming languages have been developed till now, but each language has its specific purpose. These languages vary in how they can communicate with the computer's hardware. Some programming languages can directly communicate with hardware while others have less or no access to that

hardware. Based on the accessibility of hardware, they can be classified into following categories:

1. Low-level language
2. Middle-level language
3. High-level language

1. Low-level language

The low-level language is a programming language that can directly access and communicate with the hardware, and it is represented in 0 or 1 forms, which are the machine instructions. The languages that come under this category are the Machine level language and Assembly language.

Machine-level language:

The machine-level language comes at the lowest level in the hierarchy, so it has direct access to the hardware. It cannot be easily understood by humans. The machine-level language is written in binary digits, i.e., 0 and 1. It does not require any translator as the machine code is directly executed by the computer. Machine language is the first-generation programming language.

The assembly language:

The assembly language comes above the machine language means that it has lesser access to hardware. It is easy to read, write, and maintain by humans. The assembly language is written in simple English language, so it is easily understandable by the users. In assembly language, the assembler is required to convert the assembly code into machine code. It is a second-generation programming language.

2. Middle-Level Language

Some special purpose middle-level languages were developed in the past that were used as bridge between hardware and user interaction. However, such languages have become obsolete and are not used anymore.

3. High-Level Language

The high-level languages brought revolution in programming world. They allow a programmer to write the programs which are independent of a particular type of computer. These languages are closer to human languages than machine-level languages.

High-level languages do not have direct access to the hardware therefore a translator (compiler or interpreter) is required to translate a high-level language into a low-level language.

Advantages of a high-level languages

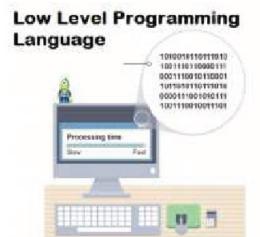
- The high-level language is easy to read, write, and maintain as it is written in English like words.
- The high-level language is portable as opposed to low-level languages; i.e., these languages are not dependent on the machine.

Differences between Low-Level language and High-Level language

Low-level language	High-level language
It is a machine-friendly language, i.e., the computer understands the machine language, which is represented in 0 or 1.	It is a user-friendly language as this language is written in simple English words, which can be easily understood by humans.
It requires the assembler to convert the assembly code into machine code.	It requires the compiler or interpreter to convert the high-level language instructions into machine code.
One type of machine code cannot run on all machines, so it is not a portable language.	The high-level code can be translated to required machine-code, so it is a portable language.
It has direct access to memory.	It is less memory efficient.
Coding and maintenance are not easy in a low-level language.	Coding and maintenance are easier in a high-level language.

- Distinguish among various types of translators

2.1.3 Translators

Computers only understand machine code (binary). This code is difficult to read, write and maintain. Programmers prefer to use a variety of high and low-level programming languages instead. A program written in any language is called as source code. To convert the source code into machine code, translators are needed.

A translator takes a program written in source language as input and converts it into a program in target machine language as output. It also detects and reports the error during translation.

Roles of translator are:

- Translating the program input into an equivalent machine language program.
- Providing alert messages wherever the programmer does not follow the rules of syntax of source language.

Different Types of Translators:

There are three different types of translators as follows:

1. Compiler

A compiler is a translator used to convert high-level programming language to low-level programming language. Compiler takes time to do its work as it translates high-level code to lower-level code all at once and creates an executable file. This translated program can be used again and again without the need for recompilation from source code.

It converts the whole program in one session and reports errors detected after the conversion. An error report is often produced after the full program has been translated. Errors in the program code may cause a computer to crash. These errors can only be fixed by changing the original source code and compiling the program again.

2. Interpreter

Interpreter is also a translator used to convert high-level programming language to low-level programming language. However, interpreter translates the code line by line and reports the error as soon as it is encountered during the translation process. With interpreter, it is easier to detect errors in source code than in a compiler. An interpreter is faster than a compiler as it immediately executes the code upon reading the code.

Interpreters do not create an executable file. Therefore, the interpreter translates the source code from the beginning every time it is executed.

3. Assembler

An assembler is a translator used to translate assembly language to machine language. It is like a compiler for the assembly language but interactive like an interpreter. An assembler translates assembly language code to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.

- Differentiate between syntax, runtime and logical errors.

2.1.4 Types of Errors

Errors are the problems or the faults that occur in the program which cause the program to behave abnormally.

Programming errors often remain undetected until the program is compiled or executed. Some of the errors prohibit the program from getting compiled or executed. Thus, errors should be removed before compiling and executing.

The most common errors can be generally classified as follows:

1. Syntax Error

Syntax error occurs when the code does not follow the syntax rules of the programming language. These can be mistakes such as misspelled keywords, a missing punctuation character, a missing bracket, or a missing closing parenthesis. Nowadays, all famous Integrated Development Environments (IDEs) detect these errors as you type and underline the faulty statements with a wavy line. If you try to execute a program that includes syntax errors, you will get error messages on your screen and the program will not be executed.

Most frequent syntax errors are:

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon

2. Run-Time Error

Errors which occur during program execution (run-time) after successful compilation are called run-time errors. A run-time error occurs when a program is asked to do something that it cannot perform, resulting in a 'crash'. The widely used example of a run time error is asking a program to divide by 0.

The code contains no syntax or logic errors but when it runs it can't perform the task that it has been programmed to carry out.

3. Logic Error

Logic errors are those errors that prevent your program from doing what you expected it to do. On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appear to be error free are called logical errors. These are one of the most common errors done by beginner programmers.

With logic errors you get no warning at all. For example, consider a program that prompts the user to enter three numbers, and then calculates and displays their average value. The programmer, however, made a logic error; one of its statements divides the sum of the three numbers by 5, and not by 3 as it should. The program will execute as usual, without any error messages, prompting the user to enter three numbers and displaying a result, but not the correct one. It is the programmer who has to find and correct the statement containing logical error.

- Discuss about Integrated Development Environment (IDE) of C++
- Develop the understanding about functions of different components of IDE

2.2 PROGRAMMING ENVIRONMENT OF C++

C++ runs on lots of platform like Windows, Linux, Unix, Mac, etc. Before we start programming with C++. We will need an environment to be set-up on our local computer to compile and run our C++ programs successfully.

2.2.1 Integrated Development Environment (IDE)

On a more basic level, IDEs provide interfaces for users to write code, organize text groups, and automate programming tools. Instead of a simple plain-text editor, IDEs combine the functionality of multiple programming processes into one. Most IDEs come with built-in translators. If any bugs or errors are found, users are shown which parts of code have problems.

Some IDEs are dedicated to a specific programming language or set of languages, having a set of tools and features which are helpful in writing codes for that language. For instance, Dev-C++ is used for making programs in C++ language. However, there are many multiple-language IDEs, such as Eclipse (C, C++, Python, Perl, PHP, Java, Ruby and more) and Visual Studio Code (Java, JavaScript, PHP, Python, Ruby, C, C++ and more).

Key Benefits of Integrated Development Environments:

- Serves as a single environment for most of a developer's needs such as compilation, linking, loading, and debugging tools.
- Code completion capabilities improve programming workflow.
- Automatically checks for errors to ensure top quality code.
- Refactoring capabilities allow developers to make comprehensive and mistake-free renaming changes.

2.2.2 Components of IDE

IDEs increase programmer productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.

Editing Source Code

This feature is a text editor designed for writing and editing source code. Source code editors are distinguished from text editors because they enhance or simplify the writing and editing of code. Writing code is an

important part of programming. IDEs facilitate this process with features like syntax highlighting and autocomplete.

Syntax Highlighting

An IDE that knows the syntax of your language can provide visual cues. Keywords, words that have special meaning like class in C++, are highlighted with different colors. Syntax highlighting makes code easier to read by visually clarifying different elements of language syntax.

Code completion

When the IDE knows your programming language, it can anticipate what you're going to type next. Code completion features assist programmers by intelligently identifying and inserting common code components. These features save developers time writing code and reduce the chances of errors.

Compiler

Compilers are components that translate programming language into a form machines can process, such as binary code. IDEs provide automated build processes for languages, so the act of compiling and executing code is done automatically.

Linker

The linker opens the compiled program file and links it with the referenced library files as needed. Unless all linker items are resolved, the process stops and returns the user to the source code file within the text editor with an error message. If no problems encountered, it saves the linked objects as an executable file.

Loader

The IDE directs the operating system's program called the loader to load the executable file into the computer's memory and have the Central Processing Unit (CPU) start processing the instructions.

Debugging

No programmer can write programs without errors. When a program does not run correctly, IDEs provide debugging tools that allow programmers to examine different variables and inspect their code step by

step. IDEs also provide hints while coding to prevent errors before compilation. Programmers and software engineers can usually test the various segments of code and identify errors before the application is released.

2.2.3 Introduction to Dev-C++

One of the most commonly used IDE for coding programs in C++ is Dev-C++. It is a graphical IDE that has an integrated compiler system to create applications for Windows as well as console. Dev-C++ is a fully featured IDE supporting features like debugging, auto completion, localization, syntax highlighting, class and variable browsing, project management, package manager and others.

Installing and Configuring Dev-C++ IDE

Dev-C++ is freely available for download from this link:
https://sourceforge.net/projects/orwelldevcpp/

After downloading the installation package, we can begin the installation process. In this book, we will be using steps for installing Dev-C++ version 5.11 with the TDM-GCC 4.9.2 compiler.

Step 1.

Select "English" as the language to be used for installation process.

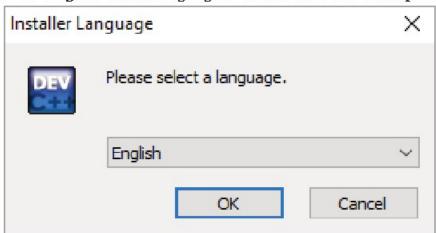



Fig. 2.1. Step 1: Dev-C++ installation

Step 2.

Agree to the license agreement by pressing “I Agree” button.

Step 3.

Select “Full” from the dropdown for “type of Install”. This will select all the necessary components required to run Dev-C++ and compile C++ source codes. Click on “Next” to proceed.

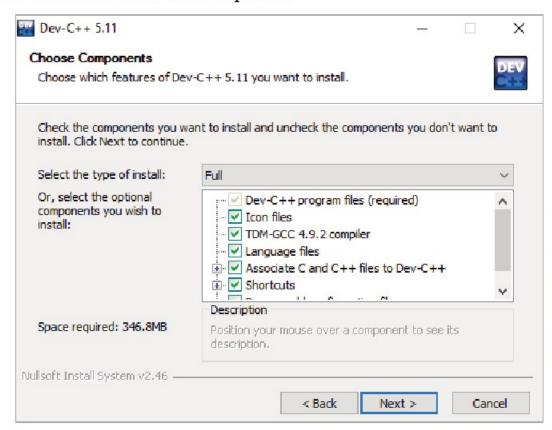



Fig. 2.2. Step 3: Installation components

Step 4.

Select the installation directory where all the necessary Dev-C++ files and libraries will be installed. Usually, the default specified path is used for installation but you can change it if desired. Click on "Install" to begin installation.

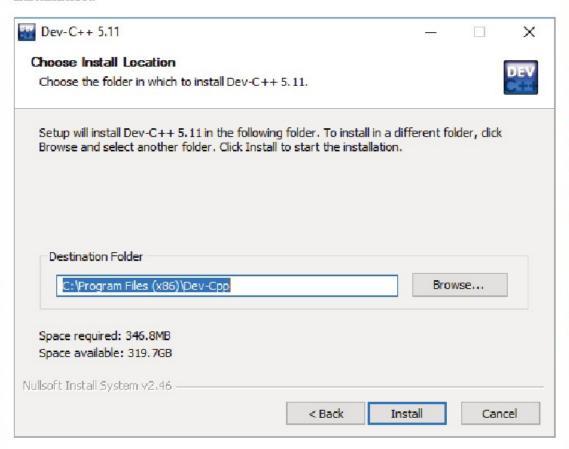



Fig. 2.3. Step 4: Install location

Step 5.

The installer will show the progress for installation. Once the process completes, it will show a "Finish" dialog. Make sure the "Run Dev-C++ 5.11" box is checked. This will automatically start Dev-C++ IDE after this installation completes. Click "Finish" button to complete the installation process.

Fig. 2.4. Step 5: Finish installation

Configuring Dev C++

When Dev-C++ IDE is run for the first time, it will require some configuration. This configuration will be used while developing programs in the IDE.

Set "English (Original)" as default interface language in the Dev-C++ first time configuration dialog. Click "Next" to continue. On the "theme" selection dialog, leave the default settings and click on "Next" to continue. Then click "OK" to close first time configuration dialog.

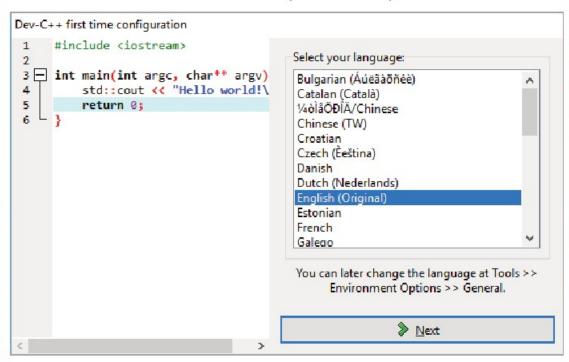



Fig. 2.5. Configuring Dev-C++

Linker Setting for Debugging

Sometimes an in-depth information is required from the debugger to properly identify the problems in our source code when a program is debugged. To obtain such information, our newly installed IDE and its integrated compiler needs to be configured. The following steps are used to enable this configuration:

1. Click on **Tools -> Compiler Options.**
2. Open the **Settings** tab from the Compiler Options dialog.
3. Under **Settings** tab, open **Linker** tab.
4. In the **Linker** tab, change the **Generate Debugging Information (-g3)** option to **Yes.**
5. Click on **OK** to save settings.

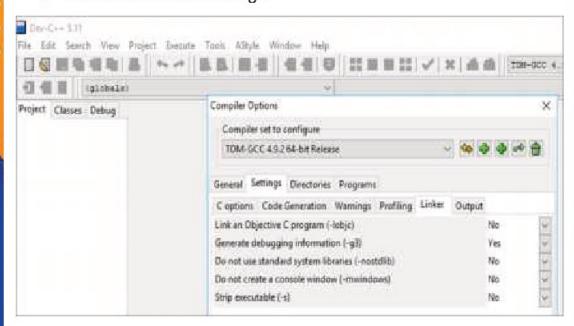



Fig. 2.6. Dev-C++ Computer options

Developing Programs in Dev-C++

C++ development is done by writing source codes and saving those files for compilation. Dev-C++ provides good project management support to help manage C++ files and group them into projects. The steps to create a new project in Dev-C++ are:

1. Click on **File -> New -> Project.**
2. From the **New Project** dialog, make sure **Empty Project** is selected. From language options, select **C++ Project**. Then enter a **Name** for your project.
3. Click on **OK**. Dev-C++ will ask for the path where you want the new project to be stored. Once it is done, Dev-C++ will open a workspace. It will show Project Explorer on the left side that shows the project we just created.

Fig. 2.7. Creating new project

Add Files to Project

A project requires source files which will contain codes for your program. The steps to create a new file are:

1. Click on **Project -> New File**. Alternatively, you can also right-click on the **Project Name** in the Project Explorer and click on **New File.**
2. Click on **Yes** on the Confirm dialog to add a file. This file is not stored until it is deliberately saved.
3. To save newly added file, click on **File -> Save**. Enter a path where you want to save the file and provide its name. Click on **Save** to store the file.

Compile and Execute Project

After writing the source codes in files, the project needs to be compiled and executed to see its output. Follow these steps to compile and run a project:

1. The project needs to be compiled before execution. To compile, click on **Execute -> Compile** or press **F9** key. **Compile Log** tab shows the compilation status. **Compiler** tab will show if there are any syntax errors.
2. After successfully compiling the project, run it by clicking on **Execute -> Run** or by pressing **F10** key.
3. A console window will open and show the output of the program.

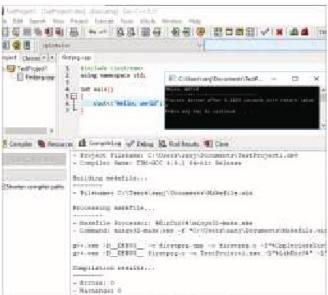



Fig. 2.8. Compile and execute project

- List out different reserved words commonly used in C++ program
- Use different data types in a C++ program

2.3 C++ PROGRAMMING LANGUAGE

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented concept. It was created by Bjarne Stroustrup and its main purpose was to make writing programs easier and more pleasant for the individual programmer.

C++ is a high-level language with an advantage of programming low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

2.3.1 Reserved Words

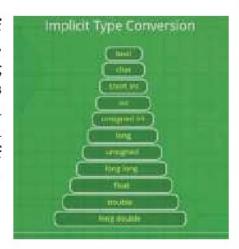
A reserved word in C++ is a word whose meaning is already defined by the compiler. A reserved word cannot be used as an identifier, such as the name of a variable, function, or label – it is "reserved from use in C++". A reserved word is part of syntax and may not have any specific meaning in English language.

There is a total of 95 reserved words in C++. The reserved words of C++ may be conveniently placed into several groups. In the first group, we put those that were also present in the C programming language and have been carried over into C++. There are 32 of these.

There are another 30 reserved words that were not in C, are therefore new to C++ programming language. Some of the commonly used C++ reserved are:

and	break	case
auto	char	do
bool	class	else
catch	const	export
default	continue	float
double	delete	goto
enum	explicit	inline
extern	false	module
for	friend	new
if	import	or
int	long	protected
nullptr	namespace	short
public	not	static
requires	operator	struct
signed	private	template
switch	register	throw
this	return	typedef
true	sizeof	union
unsigned	try	virtual
void	using	while

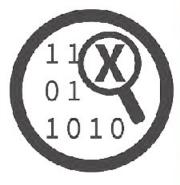
2.3.2 C++ Data Types

You may need to store information of various formats and sizes like character, integer, floating point, double floating point, boolean etc. These formats and sizes are defined as data types. Based on the data type of a storage, the operating system allocates memory and decides what kind of data can be stored in that allocated memory.

C++ offers the programmer various types of built-in as well as user defined data types. Following table lists down some of the basic C++ data types:

Type	Keyword	Size	Range
Boolean	bool	1 byte	0 (false), 1 (true)
Character	char	1 byte	-127 to 127 or 0 to 255
Integer	int	4 bytes	-2147483648 to 2147483647
Floating point	float	4 bytes	1.5×10^{-45} to 3.4×10^{38}. Stores fractional numbers. Sufficient for storing 7 decimal digits
Double floating point	double	8 bytes	5.0×10^{-345} to 1.7×10^{308}. Stores fractional numbers. Sufficient for storing 15 decimal digits

- Differentiate between variable and constant
- Comprehend variable declaration rules in C++
- Differentiate between variable declaration and initialization

2.4 CONSTANTS AND VARIABLES

A **constant** is a value that cannot be altered by the program during execution, i.e., the value is constant. When associated with an identifier, a constant is said to be "named," although the terms "constant" and "named constant" are often used interchangeably. This is contrasted with a **variable,** which is an identifier with a value that can be changed during execution.

2.4.1 Constants and Variables

A constant is a data item whose value cannot change during the program's execution. Thus, as its name implies – the value is constant. Constants are used in two ways. They are:

1. literal constant
2. defined constant

A *literal constant* is **a value** you type into your program wherever it is needed. Examples include the constants used for initializing a variable and constants used in lines of code:

```
21, 12.34, 'A', "Hello world!", false, null
```

In addition to literal constants, there are symbolic constants or named constants which are constants represented by name. The `const` keyword and `#define` preprocessor are used to define a constant. Many programming languages use ALL CAPS to define named constants like `const float PI = 3.14159;` OR `#define PI 3.14159`.

A **variable** is the memory location that can hold a value. This value can change during the program's execution. It does not remain constant. For example, a classroom with a capacity of 20 students is a fixed place or constant but the subjects taught, teachers and students will vary with each class and subject and are variables.

Variables do not require to be assigned initial values. Variables once defined may be assigned a value within the instructions of the program. Variable can be assigned different values at different times during execution. For example:

```
x = 5;
x = 37;
```

Difference between Constant and Variable:

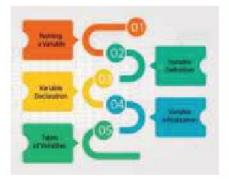
Constant	Variable
A constant does not change its value during program execution.	A variable, on the other hand, changes its value depending on instructions.
Constants are usually written in numbers and may be defined in identifiers.	Variables are always written in letters or symbols.
Constants usually represent the known values in an equation, expression or in line of programming.	Variables, on the other hand, represent the unknown values.

2.4.2 Rules for Naming Variables

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as int) cannot be used as names
- Names cannot be longer than 32 characters in C++ by default.

2.4.3 Declaring (Creating) and Initializing Variables

In C++, there are different **types** of variables (defined with different keywords). A variable declaration tells the compiler where and how much storage to create for the variable. A variable declaration specifies a data type and name for that variable as follows:

Syntax

```
data_type variable_name;
```

Where *type* is one of C++ data types (such as int), and *variable_name* is the name of the variable (such as **x** or **myName**).

Initialization

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

Syntax

```
data_type variable_name = value;
```

The **equal sign** is used to assign values to the variable.

Strings in C++

Variables that can store non-numerical values that are longer than one single character are known as strings.

The C++ language library provides support for strings through the standard string class. This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage. Strings can be declared without an initial value and can be assigned values during execution.

- A computer program is a list of instructions that tell a computer what to do.
- We refer to syntax in computer programming as the concept of giving specific word sets in specific orders to computers so that they do what we want them to do.
- Different programming languages can be classified into high, middle and low-level languages.
- High-level languages are easy to read for humans and contain English language like words.
- Middle-level languages have a human readable format along with direct control over the machine's resources.
- Low-level languages are easy for machines to read and hard for humans. Low-level programs mostly comprise of binary digits and memory operators.
- There are three types of translators namely, compilers, interpreters and assemblers.
- Compilers convert high-level languages into machine readable format.
- Interpreters also convert high-level programs into machine readable format.

- Unlike compilers, interpreters convert instructions line-by-line.
- Assemblers convert low-level languages into machine readable format with added benefit of being interactive like an interpreter.
- Programming errors prevent the program from being compiled or executed.
- Syntax errors are words or symbols unrecognized by a particular programming language.
- Runtime errors only occur during program execution mostly due to an invalid input.
- Logical errors are considered when incorrect results are obtained based on provided input.
- Logical errors do not interrupt program execution.
- Integrated Development Environments (IDEs) are programs that facilitate writing, compiling and executing codes.
- IDEs usually provide a single environment for programmers to write and executes codes efficiently.
- C++ is a general-purpose high-level programming language.
- Reserved words are part of programming language syntax and cannot be used as name of variable, function or label.
- A constant is a named identifier having a value that cannot be changed.
- A variable is a named identifier with a value that can be changed during normal execution of program.
- Different types of values can be stored in variables. These types are called data types such as int, string, bool, etc.
- A variable can be declared by giving it a name and type. It can also be initialized during declaration by assigning a value to it.
- In C++, a variable is defined and initialized as:

  ```
  “data_type variable_name= value;”
  ```

- C++ offers various data types for holding values in variables.
- These data types allocate system memory based on its type.

A. ENCIRCLE THE CORRECT ANSWER:

1. A computer program is a collection of:
   a. Tasks b. Instructions
   c. Computers d. Programmers
2. High-level languages have syntax that is:
   a. Easily readable by humans b. Easily readable by machines
   c. Easily readable by both d. None of the above
3. Low-level languages have syntax that is:
   a. Easily readable by humans b. Easily readable by machines
   c. Easily readable by both d. None of the above
4. The primary characteristic of a compiler is to:
   a. Translate codes line-by-line
   b. Translate low-level code to machine language
   c. Detect logical errors
   d. Translate codes all at once
5. The primary characteristic of an interpreter is to:
   a. Translate codes line-by-line
   b. Translate low-level code to machine language
   c. Detect logical errors
   d. Translate codes all at once
6. An Integrated Development Environment facilitates a programmer to:
   a. Edit source code b. Complete and highlight syntaxes
   c. Debug and compile codes d. All of the above
7. All errors, detected by users are typically:
   a. Syntax Errors b. Semantic Errors
   c. Run- Time Errors d. Logical Errors
8. Allowed names for declaring a variable:
   a. Can contain whitespaces
   b. Can be one of the reserved words
   c. Can contain letters, digits and underscores
   d. Can be the same as its data type

9. A bool data can store following type of value:
   a. Numbers b. Strings
   c. Fractional numbers c. True or false
10. Which data type occupies the most space in memory?
   a. Character b. Integer
   c. Floating point d. Double floating point

B. RESPOND THE FOLLOWING:

1. What is computer program?
2. List five common high-level languages used and describe their purpose.
3. Using the rules of naming variable, develop ten meaningful and valid variable names.
4. Write and two differences between machine and assembly language..
5. What are Strings in C++?
6. What is the difference between declaring and initializing a variable?
7. What is the difference between source code and object code?
8. List any four advantages of using an IDE.

LAB ACTIVITIES

1. In groups, students should learn to download, install and configure Dev C++.
2. Teacher demonstrates the use of IDE and its features as given in this unit. Also explains the use of variable and constants.

Unit 3

INPUT/OUTPUT HANDLING IN C++

- Explain basic structure of C++ program.
- Introduce the use of Preprocessor directives in C++ program.
- Comment Statement in C++

3.1 BASIC STRUCTURE OF C++

In C++ program is divided into three parts:

1. Preprocessor Directives
2. Main Function Header
3. Body of Program / Function

The basic structure of the C++ program is given in Figure No. 2.1:

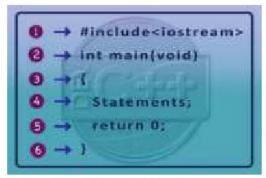



Fig. 2.1 Basic Structure of C++ Program

S.NO.	Codes and Symbols	Description
1.	#include<iostream>	The # symbol is called Preprocessor Directives. `#include` is used to link the external header files/libraries which may be required in program. `#define` is used define constants in program.
2.	using namespace std;	This instruction tells the compiler to use standard namespace. The Namespace is the collection of identifiers. It is used for variables, class, functions, and objects. All these elements of the Standard Library of C++ are declared within standard **"std"**.
3.	int main(void)	`int main(void)` function is used for execution of C++ program. void main(void) nothing to return value.

4.	{	This symbol indicates the beginning of the main function. It is also known as Opening Curly braces.
5.	Statement;	Instructions that perform a particular task is called a statement. Statement terminator (;) used to the end of statements. This symbol also known as semi colon. For example: **`cout << "Pakistan Zindabad";`** The output of the given example is that "Pakistan Zindabad" will print on the screen.
6.	return value;	The return value is the exit code of your program. By default, **`main( )`** in C++ returns an **`int`** integer data type value to the operating system.
7.	}	This symbol indicates the ending of the main function. It is also known as Closing Curly braces.

The body of the function is enclosed between curly braces. All instructions are executed within opening "{" and closing "}" curly braces.

3.2 COMMENT STATEMENT IN C++

The comment statement are those statements that are ignored by the compiler. These statements do not execute. Through comments, the programmers give special remarks to the statements for their convenience. In C++, there are two types of comment statements.

1. Single Line Comment
2. Multi Line Comment

1. Single Line Comment:

It is used to a single-line explanation with the help of a double slash (//) symbol. If the programmer wants to use a single line comment on more

than one line, this may need to put a double slash on each line at the start. These comments are ignored by the compiler, which means comments are not executable.

For example:

```
// Single line comment
// This is my first program
#include<iostream>
int main( )
{
      Statements…;
return 0;
}
```

2. Multi Line Comment:

It is used for multiple-line explanations. Symbols (/* and */) are needed at the start and end of the statements. These comments are ignored by the compiler, which means comments are not executable.

For example:

```
/* Multi line comment
  This is my first program */
#include<iostream>
int main( )
{
      Statements…
return 0;
}
```

- Differentiate between input and output functions.
- Use input and output functions in a program.
- Describe the use of statement terminator in a program
- Use escape sequences in any C++ program.

3.3 INPUT/OUTPUT Handling in C++

In C++Input and Output streams perform Input/ Output (I/O) operation and these I/O stream are stored in header file. Such as <iostream>. These header files must be mentioned at the beginning of the program.

I/O streaming uses multiple input/output channels.

3.3.1 Output Function

S.no.	Functions	Description with examples
1.	cout statement	**cout** is a predefined object in C++. It is used to display the output to the standard output device i.e. monitor. "cout" uses insertion operator (<<). Syntax: cout << variable or cout <<exp./string << variable **For example:** `cout << "MY FIRST PROGRAM";`

2.	puts( )	This function used to print the string to the output stream. The new line is automatically inserted after printing the string.
		Syntax: int puts(const char * str);
		For example :
		`#include<iostream>`
		`using namespace std;`
		`int main(void)`
		`{`
		`puts("MY FIRST PROGRAM");`
		`return 0;`
		`}`
		OUTPUT:
		MY FIRST PROGRAM

"**cout**" stands for "**Character Output**". Here C means character and Out means output.
puts( ) is defined in <cstdio> header file. This file must be included at the beginning of the program.

Teachers Note

Teacher are supposed to orient students also about the function of **putc( )** in C++ program.

3.3.2 Input Function

S.no.	Functions	Description with examples

1.	cin statement	**cin** is a predefined object that reads data from the keyboard with the extraction operator (>>). This operator allows you to accept data from standard input device.
		Syntax: cin >> variable;
		For Example:
		`#include<iostream>`
		`using namespace std;`
		`int main(void)`
		`  {`
		`   int a;`
		`   cin >> a;   //cin takes input in "a"`
		`variable`
		`   return 0;`
		`  }`

2.	getch( )	• **getch( )** is predefined function. This function is defined in conio.h (Console Input and Output header file.
		• It is used to get a single character from keyboard during execution of program.
		• The entered character is not printed on the screen.
		• It is used to hold the output screen until the user press any key from the keyboard.
		For Example:
		`#include<iostream>`
		`#include<conio.h>`
		`using namespace std;`
		`int main(void)`
		`{`
		`  char ch = getch();`
		`  cout <<"X Class";`
		`  cout << ch;`
		
		`  return 0;`
		`}`

3.	getche( )	• The function of "getche( )" is similar to getch( ) function.
		• The "**getche( )**" stands for get character echo.
		• This function displays the character that entered by the user.
		• It is also predefined function in "conio.h" header file.
		Syntax: character variable = getche( );
		For example:
		`#include<iostream>`
		`#include<conio.h>`
		`using namespace std;`
		`int main(void)`
		`{`
		`char ch;`
		`int a=10,b=10;`
		`cout <<"\n Do you want to continue (Y/N)...";`
		`ch=getche();`
		`cout << "\n the addition is...." <<a+b;`
		`return 0;`
		`}`
		OUTPUT:
		Do your want to continue(Y/N)..Y
		the addition is..... 20

4.	getchar( )	The getchar( ) function in C++ reads the character from standard input stream. It's defined in <stdio.h> header file. It needs to press Enter Key after entering the character.
		For example:
		`#include<iostream.h>`
		`#include<stdio.h>`
		`using namespace std;`
		`int main(void)`
		`{`
		`char ch;`
		`cout << "\n Use of getchar`
		OUTPUT:
		Use of getchar function....a
		getchar is.......a

		function";
		 ch = getchar();
		 cout << "\n getchar is" << ch;
		
		 return 0;
		 }

5.	gets( )	This is a predefined function in C++ and it reads characters from stdin and stores them until a newline character found. It's defined in <cstdio> header file. Program's code shows how to apply this function in C++.
		 Syntax: gets(variable);
		
		 For example:
		 #include<iostream>
		 #include<cstdio.h>
		 using namespace std;
		 int main(void)
		 {
		 char ch[20];
		 cout << "\n Enter the message..";
		 gets(ch);
		 cout << " Your message is…." << ch;
		 return 0;
		 }
		
		 OUTPUT:
		 Enter the message….
		 Pakistan
		 Your message is…….
		 Pakistan

Teachers Note

It is good if teacher informs students about the function of **getc( )** in C++

3.3.3 Statement Terminator (;):

Every statement in C++ must be terminated with semi colon (;). It indicates the end of the statement and it is also called Statement terminator. If the terminator is missing an error message will occur.

3.3.4 Escape Sequences:

The escape sequences are special non-printing characters. They can be used with the "cout" in C++. An escape sequence starts with a backslash (\) and a code character.

The commonly used escape sequences are given below:

Escape Sequence	Explanation with example
\a	"a" means Alert or alarm. It causes a beep sound in the computer. **Example:** `cout <<"\a";`
\b	"b" stands for backspace. It moves the cursor backspace. **Example**: `cout << "\b";`
\t	"t" stands for Horizontal tab. It is used to shift the cursor to a couple of spaces to the right in the same line. **Example:** `cout << "\t";`
\n	"n" stands for New line or line feed. It inserts a new line and cursor moves to the beginning of the next line. **Example:** `cout << "\n";`
\r	Carriage Return "r". It is used to position the cursor to the beginning of the current line. **Example:** `cout << "\r";`
\\	"\ backslash" It is used to print the backslash character. **Example**: `cout << "\\t";`
\'	"Single quotation" It is used to print the "apostrophe (') sign or character. **Example**: `cout << "\'";`
\"	"Double quotation" It is used to print the quotation mark. **Example**: `cout << \";`

Teachers Note	Teachers should ask students to develop program using escape sequences.

3.4 OPERATORS:

Operators are special symbols used for specific purposes. Operators perform mathematical operations on Operands. For example: x + y. Here "x" and "y" are operand and "+" operator. There are seven types of operators that are used in C++ programming.

Operator
x + y
Operands

- 3.4.1 Arithmetic Operators
- 3.4.2 Increment Operators
- 3.4.3 Decrement Operators
- 3.4.4 Relational Operators
- 3.4.5 Logical Operators
- 3.4.6 Assignment Operators
- 3.4.7 Arithmetic Assignment Operators

3.4.1 Arithmetic Operators:

In Arithmetic operators, five different operators are used to perform an arithmetic operation. All operators except Remainder or Modulus operator can be used in integer and float data type.

- **Addition (+):** It is used to perform arithmetic addition.

  Example: a + b;
- **Subtraction (-):** It's used to perform arithmetic subtraction.

  Example: a – b;
- **Multiplication (*):** It used to perform arithmetic multiplication.

  Example: a * b;
- **Division (/):** It perform the arithmetic division of two numbers.

  Example: a / b.
- **Remainder / Modulus (%):** It used to find remainder of a division. It returns the remainder of an integer value. Remainder operator is also known as Modulus operator. This operator is used only with integral data types.

  Example: 5/2 = 2 and 1 is remainder. We can write in this form 5%2

Develop a simple calculator in C++ by using arithmetic operators.

```
#include<iostream>
#include<cstdio.h>
using namespace std;
int main(void)
{
      int a,b,add,sub,multi,remd;
      float div;
      cout <<"\n \t CALCULATOR";
      cout <<"\n \t ============";
      cout << "\n \t Enter the Value of a........";
      cin >> a;
      cout << "\n \t Enter the value of b........";
      cin >> b;
      add = a+b;
      cout << "\n \t Addition of "<< a <<"and.. "<< b << "is...."<<
add;
      sub=a-b;
      cout << "\n \t Subtraction of "<< a <<"and..."<< b <<
"is...."<< sub;
      multi=a*b;
      cout << "\n \t Multiplication of "<< a <<"and..."<< b <<
"is...."<< multi;
      div=a/b;
      cout << "\n \t Division of "<< a <<"and..."<< b << "is...."<<
div;
      remd=a%b;
      cout << "\n \t Remainder of Modulus division of "<< a
<<"and..."<< b << "is...."<< remd;
return 0;
}
```

OUTPUT

CALCULATOR

========

Enter the Value of a..... 15

Enter the Value of b..... 10

Addition of 15 and 10 is 25

Subtraction of 15 and 10 is...... 5

Multiplication of 15 and 10 is 150

Division of 15 and 10 is...... 1

Remainder of Modulus division of 15 and 10 is.....5

In the division of integers, numbers show only whole numbers in result.

Teachers are supposed to explain the precedence of Arithmetic Operators.

3.4.2 Increment Operator (++):

The increment operator can be used with any type of variable. It is used to add 1 to the value of a variable. Increment operator represented by ++ (double plus sign). This operator can be applied only to a single variable. There are two ways to use increment operator:

- Prefix Increment Operator: You can apply this operator before the variable name. It can be written like **++a**. i.e. **x=++a;**

Example:

```
#include<iostream>
using namespace std;
int main(void)
{
int a=10;
cout << "\n Value of a is....."<< ++a;  ⟹ Prefix Increment Operator
return 0;
    }
```

Output:
Value of a is 11

In prefix increment operation value of a is printed as 11 because 1 is added in 'a' before printing.

- Postfix Increment Operator: If the increment operator is applied after the variable name, it is known as Postfix Increment operator. It is written like **a++**.

Example:

```
#include<iostream>
using namespace std;
int main(void)
{
int a=10;
cout << "\n Value of a is....."<< a++;  ⟹ Postfix Increment Operator
return 0;
}
```

Output:
Value of a is 10

In Postfix increment operation value of a is printed as 10 because 1 is added in 'a' after printing. So the value of 'a' will change to 11 after printing.

3.4.3 Decrement Operator (--):

The decrement operator is same as increment operator but it subtracts 1 from the value of a variable. It is represented by -- (double minus sign). It can also be used as prefix and postfix.

3.4.4 Relational Operator:

The relational operators are used to test the relation between two values. All relational operators are binary operators. These operators must require two operands. The result of the comparison is **True (1)** or **False (0).** The relational operators are also known as Comparison Operators.

The following are the relational operators and their operations.

Operators	Meaning	Purpose	Expression
==	Equal to	Its check the equality of two operands values.	a==b
!=	Not equal to	It checks whether the value of the left operand is not equal to the value of the right operand.	a != b

>	Greater than	This operator checks the value of left operand is greater than the value of right operand.	a > b
<	Less than	Its check the value of left operand is less than the value of right operand.	a < b
>=	Greater than or equal to	It checks the value of the left operand is greater than or equal to the value of right operand.	a >= b
<=	Less than or equal to	It checks the value of the left operand is less than or equal to the value of right operand.	a <= b

Use relational operators in C++ program:

```
// Relational Operators
#include<iostream>
using namespace std;
int main(void)
{
    int a = 10;
    int b = 20;
    cout << "\n \t Relational Operator";
    cout  < "\n \t ===========";
    cout  <<  "\n  \t"  <<  "False  \t"<<  (a  ==  b)<<"\t  false
because 10 is not equal to 20";
    cout << "\n \t" << "True \t"<< (a < b) <<"\t true because
10 is less than 20";
    cout << "\n \t" << "False \t"<<(a > b) <<"\t false because
10 is not greater than 20";
    cout << "\n \t" << "False \t"<<(b <= a)<<"\t false because
20 is not less than or equal to 10";
    cout << "\n \t" <<"True \t"<< (a>=a)<<"\t true because 10
is not greater than 10 but equal to 10";
    cout << "\n \t" << "True \t"<<(b!=a)<<"\t true because 20
is not equal to 10.
return 0;
}
```

OUTPUT

```
0
1
0
0
1
1
```

3.4.5 Logical Operator:

Logical operators are used to determine two relational expression. These operators can be used in many conditional and relational expressions. There are three logical operators that are used in C++ programming.

Operators	Description	Expression
&&	This operator is called AND. The condition will be true if both expressions are true.	`x=10, y=5, z=12` `x>y && x<z`
||	It's known as OR operator. The condition will be true if any one of the expressions is true.	`x=10, y=5, z=12` `x>y || x>z`
!	This operator is called NOT. The condition will be inverted, false becomes true and true becomes false.	`x=10, y=5;` `!(x<y);`

Differentiate between relational and logical operators

Relational Operator:

- These operators are used to perform logical operations on two given variables.
- The relational operators are used to compare any two values.

Logical Operator:

- These operators are used to compare the two relational statements.
- The logical operators are used to combine one and more than one relational expression.
- Like relational operators they also give **True (1)** or **False (0) results.**

Use Logical and Relational operators in C++ program:

```
   #include <iostream>
   using namespace std;
   int main(void)
   {
   int x = 10;
   int y = 5;
   int z = 12;
   cout << "\n \t LOGICAL OPERATOR";
   cout << "\n \t ================";
   cout << "\n \t" << ((x > y) && (x < z)) << "\n \t ADD
     OPERATOR"<< endl;
   cout << "\n \t" << ((x > y) || (x > z)) << "\n \t OR
OPERATOR"<< endl;
   cout << "\n \t" << !(x < y) << "\n \t NOT OPERATOR" <<
endl;
   return 0;
   }
```

OUTPUT

```
1
1
1
```

3.4.6 Assignment operator (=) vs Equal to operator (==)

Assignment operator (=)	Equal to operator (==)
The assignment operator (=) is used for assigning a variable to a value.	The equal to (==) operator is used to check the equality of two operands values.
This operator assigns the value of right-side expression to left-side variable. Such as x=10;	This operator compares value of the left side and right-side expression. Such as x=10 and y=10 than x==y If condition true otherwise false.

Use assignment operator in initialization of variable and equal to operator in order to compare two variables.

```
#include<iostream>
using namespace std;
int main(void)
{
      int x=20, y=10;
      cout << "\n \t Assignment vs Equal to Operator";

      cout << "\n \t ================;
      cout << "\n \t x = 20 assignment opt....." << x;
      cout << "\n \t y = 10 assignment opt....." << y;
      cout << "\n \t Equal to opt. result is....." <<
(x==y);
return 0;
}
```

OUTPUT

x=20 assignment opt.....20
y=10 assignment opt.....10
Equal to opt is.......0

Arithmetic Assignment Operators:

In arithmetic assignment operators, the arithmetic operator is combined with the assignment operator. The assignment operator comes to the right of an arithmetic operator. This operator is also known as Compound Assignment Operator.

Operators	Description
+= (Addition-assignment)	It adds the right operand to the left operand and assigns the result to the left operand. Example: a+=2 means a=a+2
-= (Subtraction-assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand. Example: a -=3 means a=a-3
*= (Multiplication-assignment)	It multiplies right operand with the left operand and assigns the result to the left operand. Example: a *=4 means a=a*4
/= (Division-assignment)	It divides left operand with the right operand and assigns the result to the left operand. Example: a /=4 means a=a /4

- The C++ program consiss of three parts.
    - Preprocessor Directives
    - main Function header
    - Body of program
- #include<iostream> is used to include header files like iostream.h, conio.h, etc.
- namespace is the collection of identifiers.
- The main( ) function is compulsory element of the C++ program.
- The comment statements are those statements that are ignored by the compiler. These statements are not executable.
- I/O Stream is a standard library file that contains definitions of Standard Input and Output functions.
- cout is an output object. It is used to display the output through output device like monitor.
- puts() is a string function and it is included in <cstdio> header file.
- cin works as an input object in C++.
- Statement terminator (;) is used for statement endngin C++ programming.
- Escape Sequences is a non – printable characters. It is used only with cout statement.
- Operators are special symbols used for specific purposes.
    - Arithmetic Operator are used for arithmetic operations or calculation.
    - Increment and Decrement Operator are used in two different ways in programming.
        - Prefix
        - Postfix
    - Relational Operators are used to test the relation between two values.
    - Logical Operators are used to determine two relational expression. Relational and Logical Operators works on a decision making and loops.

A. ENCIRCLE THE CORRECT ANSWER:

i) The C++ header file ____________ contains function prototype for the standard Input and Output functions.

a. <iomain.h> b. <iostream>
c. <fstream.h> d. <cstdio.h>

ii) Which operator is used for input stream?

a. > b. <<
c. >> d. <

iii) gets stands for ____________________.

a. get stream b. get string
c. get str d. get std.

iv) getch() and getche() are included in _____________ header file.

a. <cstdio.h> b. <conio.h>
c. <stdlib.h> d. <stdio.h>

v) Which operator is used for logical AND operation?

a. & b. &&
c. | | d. !

vi) Which of the following operator is correct to compare two values of variables?

a. = b. <=
c. == d. both b and c

vii) Which of the following needs pressing Enter Key from the keyboard?

a. getch( ) b. getche( )
c. getchar( ) d. gets( )

viii) != operator belong to which type of operator.

a. Relational b. Logical
c. Arithmetic d. None of these

ix) Which operator add the first operand to the second operand and gives the result to first operand.

a. *=   b. +=

c. ++   d. +

x) `cout << 12-6/2;` What will be the result on screen?

a. 3   c. 6

c. 9   d. 12

B. RESPOND THE FOLLOWING:

1. Use \a and \r both escape sequences in a program.
2. How many types of **comment** statements are used in C++?
3. Differentiate between Arithmetic operators and Relational operators.
4. Write a program in C++ and use all arithmetic assignment operators.
5. What is basic difference between Assignment operator and Equal to operator.
6. What is the basic difference between \n and \t ?
7. Get the output of following program.

```
#include <iostream>
using namespace std;
int main(void)
{
    int a = 27;
    cout << "a is " << a << endl;
    cout << "a is now" << a++ << endl;
    cout << "a is now " << a << endl;
    cout << "a is now " << --a << endl;
    cout << "a is now " << a << endl;
return 0;
}
```

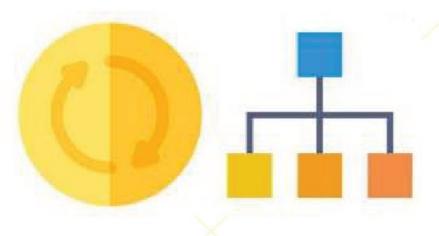
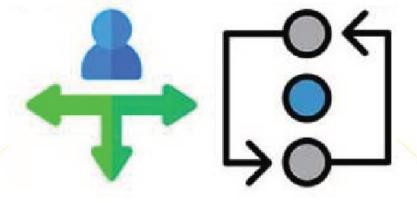
LAB ACTIVITIES

1. Develop programs for manipulating the following formulas.

Title	Formula	Description
Calculating Speed of an Object	s=d/t	Speed = distance / time
Newton's second law of motion	F=ma	
Calculating acceleration	a=(vf-vi)/t	Acceleration is equal to (final v - initial v) / time
Area of Triangle	a= ½ bh	area = (1/2) (base) (height)
Convert the Celsius to Fahrenheit	F=(c*1.8)+32	

2. Write a program to calculate the volume of a box.
3. Write a program of Marksheet takes input of five subjects, print its total and percentage also.
4. Write a code to calculate mathematical expression of $a^2+2ab+b^2$.
5. List out errors from the C++ program and remove those errors, write the output.

```
#include<iostream.h>
using namespace std
int main(void);
{
  int x ;
  cout << "\n Enter the value of x……";
  cin >> x
  cout << "\n The square of x………" << a * a ;
return 0;
}
```

CONTROL STRUCTURE

- Recognize the various types of control statements:
- Define decision making structure
- Understand the syntax of If and If-Else statements
- Use If and If-Else statements in C++ programming
- Differentiate between If, If-Else and switch decision making structures
- Use switch statement in the programs

4.1 CONTROL STATEMENTS

A computer program is the set of instructions in sequential form. These instructions execute from top to bottom. We can control the flow of program with the help of Control Statements. Control Statements are used to control the direction of program by repeating any block of code, making a choice or simply transfer the control to any specific block of program. These statements help programmers to decide which part of code is executed at certain time.

C++ has three types of control statements:

1. Selection/Decision Making Structure
2. Iteration / Loops
3. Jump

4.2 SELECTION/DECISION MAKING STRUCTURE

They are used to decide whether a certain part of code is executed or not.

C++ has three decision making structures;

1 'if' statement

2 'if- else' statement

3 'switch' statement

4.2.1 "if" statement

It is the basic decision statement. The structure contains "if" keyword followed by a conditional expression in parenthesis and then its body of statements(s) also called if block. If there are more than one statement in "if" block or body we enclose all of them in braces.

If statement checks a condition, if the condition is true the statements in "if" block are executed and if condition is false it leaves the statements in "if" block and starts executing statements after "if" block.

Fig: 4.1: if Statement's Flowchart

Syntax:

```
    if (test condition)
       {
        Statement(s);
```

Following is an example of using if in a program. This program takes marks as input. If marks are greater than 60 then it adds word "good" between "you are a " and "student of this class

```
/* Program 1.   "if" statement example */
// writes good on basis of marks
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
      int marks;
      cout<<"\nEnter Marks ";
      cin>>marks;

      cout<<" You have secured "<<marks<< " marks ";

      if(marks>=70)
                {
            cout<<" and A Grade ";
                }
      cout<<" in 5th class";
return 0;
}
```

Nested "if" Statement

An 'if' statement which is part of block of another 'if' statement is called nested 'if'. The inner 'if' statement will only be tested if the outer 'if' is true.

In following program, marks and age of a person are taken as input. First condition checks marks, if they are greater than or equal to 60 then next condition checks age. If age is also greater than 18 then it prints message "you got the job". "good luck" is always printed.

```
/* Program 2.    Nested "if" statement example   */
// job given message on basis of marks and age
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
      int marks,age;

      cout<<"\nEnter your Marks ";
      cin>>marks;
      cout<<"\nEnter your age ";
      cin>>age;

      if(marks>=60)
            {
            if(age>=18)
                  {
                  cout<<"you got the job";
                  }
            }
      cout<<" Good Luck";
return 0;
}
```

4.2.2 "if-else" statement

The structure of if- else contains 'if' keyword followed by a conditional expression in parenthesis and then its body of statements(s) then 'else' keyword and its body of statements. It checks the condition, if the condition is true the statements in if block are executed and in case of false condition, it executes statements in 'else' block. It means either 'if block statements or 'else' block statements must execute.

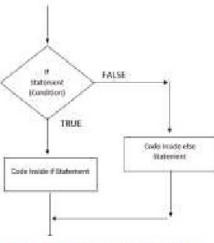



Fig: 4.2. if- else Statement's Flowchart

Syntax:

```
If (test condition)
{
        Statement(s);
}
Else
{
    Statement(s);
}
```

For example, following program takes marks as input and decides pass or fail on the basis of marks. If greater than or equal to 40 then pass otherwise fail.

```
/* Program 3. "if-else" statement example */
// shows pass or fail on the basis of marks
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
        int marks;
        cout<<"\nEnter your Marks ";
        cin>>marks;

        if(marks>=40)
                cout<<"You are pass";
        else
                cout<<" You are fail";
return 0;
}
```

4.2.3 else-if statement

In "if" statement if we use nesting deeply i.e. one if is nested to other and other is nested in another and so on then in indentation we see a ladder like structure which is difficult to understand like below. Here is an example. This program takes marks as input and then determines the grade by applying if condition multiple times.

```
/* Program 4. "else-if" statement example */
// shows Grade on the basis of marks
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
      int marks;

      cout<<"\nEnter your Marks ";
      cin>>marks;

      if(marks>=80)
            cout<<"Grade is A1";
      else
            if(marks>=70)
                     cout<<" Grade is A";
                       else
                         if(marks>=60)
                            cout<<" Grade is B";
                              else
                                 if(marks>=50)
                                     cout<<" Grade is C";
                                          else
                                              cout<<" Fail";
return 0;
}
```

However, to make this program look simpler and more understandable, we may design it in another format by using "else-if" statement. This format only increases the readability of program while there is no any change for compiler. "else-if" statement for the above program is shown in next program where all "else" are in a single column.

```
 if(marks>=80)
      cout<<"Grade is A1";
else if(marks>=70)
                cout<<" Grade is A";
      else if(marks>=60)
                cout<<" Grade is B";
      else if(marks>=50)
                 cout<<" Grade is C";
      else
                  cout<<" Fail";\
```

Now, make a program in which user will input two integers and one arithmetic operator (+, -, *, /). Perform the given arithmetic operation on given numbers by using else- if statement and print the result on screen.

4.2.4 "switch" statement

The switch statement starts with **"switch"** keyword followed by a variable or expression in parenthesis, then a block of switch statement in braces. The block contains one or more case statement each followed by integer or character constant and then colon. After colon there may be many statements followed by "break" statement in the end. The switch variable or expression checks its value equal to these constants that follow case statement. If it matches to any of them then control is transferred to that **"case"** and all statements after colon are executed and switch is broken with "**break**" statement and control transfers to the next

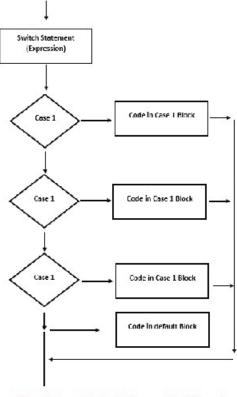



Fig: 4.3. switch Statement's Flowchart

statement after switch. If switch variable does not match with any of the case constants control goes to keyword **"default"** (if it is present). It acts as **"else"**. **"break"** and **"default"** keywords are optional.

Syntax:

```
switch(expression)

    {
    case constant 1:
       statement(s)
       break;
    case constant 2:
       statement(s)
    break;

   .    .    .
   .    .    .
    default:
       statement(s)
    }
```

In following example, the program takes number of the day of week (from 1 to 7) as input and on the basis of case matching prints the name of the day. If other number is given then invalid day message is shown through default statement.

```
/* Program 5.   "switch" statement example */
// prints name of day checking day number
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
     int dow;

     cout<<"\nEnter number of weekday 1 to 7 ";
     cin>>dow;

     switch(dow)
```

```
            {
            case 1: cout<<"Sunday"; break;
            case 2: cout<<"Monday"; break;
            case 3: cout<<"Tuesday"; break;
            case 4: cout<<"Wednesday"; break;
            case 5: cout<<"Thursday"; break;
            case 6: cout<<"Friday"; break;
            case 7: cout<<"Saturday"; break;
            default: cout<<"Invalid day number";
            }

return 0;
}
```

Can you make a program to input month's number and print its name on screen?

4.2.5 Difference between If, If-Else and switch decision making structures

"if"	**"if"** statement checks a condition using relational and other operators, if the condition is true the statements in if block execute. If condition is false, it leaves the statements in **"if"** block and starts executing statements after **"if"** block.
"if-else"	**"if-else"** statement checks a condition using relational and other operators, if the condition is true the statements in **"if"** block are executed and in case of false condition it executes statements in **'else'** block. It means either **"if"** block statements will execute or **"else"** block statements.
"switch"	**"switch"** statement checks 'switch' variable value equal to constant that follows case statement. If it matches to any of them then control is transferred to that case and all statement after colon are executed and switch is broken with **"break"** statement. Otherwise, control is transferred to **"default"** statement. It has some limitations. It only matches character and integer data type variables, it checks switch variable value only with case constants not with any variable and it also cannot use relational operators like less than (<) or greater than (>) and exactly matches value with case constants.

- Explain the concept of loop structure
- Explain for, while and do-while loop structures
- Differentiate between for, while and do-while loop structures and their use
- Use these three loop structures into C++ programming
- Explain the concept of nested loops

4.3 ITERATION/ LOOP

Normally statements are executed sequentially, one after the other. In some situations, we need to execute a block of statements several number of times. Loops allow us to execute a statement or a group of statements several numbers of times. A group or block is made by enclosing statements in braces. There are three types of loops in C++.

```
1. for
2. while
3. do- while
```

4.3.1 "for" loop

This loop executes a sequence of statements multiple times. It is usually used in situations where at the start of loop we know that how many times loop block will execute. It is a pre-test loop as condition is tested at the start of loop. It starts with keyword **"for"** followed by loop expression in parenthesis. Loop expression has three parts separated by semicolon.

Fig: 4.4. for Loop Statement's Flowchart

First part is initialization which initializes loop variable; second part is test expression which tests loops variable and third part in increment or decrement in loop variable. Then comes the body of the loop which may have one or more statements. If there are more than one statements then they are enclosed in braces.

When loop starts, first part is executed, only one time. Then second part tests the condition, if it is true it enters the loop otherwise transfers control to statement after loop. If it enters in loop then after executing all statements in loop block, third part is executed and again condition is tested. It is also called counter controlled loop or definite repetition loop since the number of iterations is known before loop execution.

Syntax:

```
for (initialization; condition testing; increment/decrement)
{
   statement(s);
}
```

The following program shows even numbers from two to twenty. It initializes loop variable count with two and then print numbers by adding two each time to this variable until count variable remains less than twenty.

```
/* Program 6. "for" loop example */
// shows even numbers from 1 to 20
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
      int count;


      for(count=2;count<=20;count=count+2)
            {
            cout<<"\n Number= ";
            cout<<count;
            }
return 0;
}
```

4.3.2 "while" loop

It is a pre-test loop. It tests the condition at the start of loop before executing the body of loop. It is usually used in situations where we do not know at the start of loop, how many times loop block will execute. So, it is also called indefinite repetition loop. It starts with keyword "while" followed by a conditional expression like if statement in parenthesis. Then comes the body of the loop which may have one or more statements in braces.

Fig: 4.5. while Loop Statement's

When loop starts, it tests a condition, if it is true it enters in the loop otherwise transfers control to statement after loop. If it enters in loop then after executing all statements in loop block again condition is tested.

Syntax:

```
[illegible]
{
    [illegible]
}
```

Following program takes characters as input through getche() in loop. When user presses enter key equal to '\r' in C++ loop condition is false and it shows total number of typed characters.

```
/* Program 7. "while" loop example */
// counts number of characters typed
#include<stdio.h>
#include <iostream.h>
#include<conio.h>
using namespace std;
int main(void)
{
	int num=0;
	char ch;

	clrscr();
	cout<<"Type any word or text, Press enter to terminate -> ";
```

```
       ch=getche();
       while(ch!='\r')
             {
             num++;
             ch=getche();
             }
       cout<<"\n Total number of characters typed "<<num;
return 0;
}
```

4.3.3 "do while" loop

It is similar to while loop, except that it tests the condition at the end of the loop body and so it is also called post-test loop. Its statements block is executed at least once. For second time condition is tested. It starts with keyword "do" followed by a body of loop which may have one or more statements in braces. Then "while" keyword and conditional expression in parenthesis. It must be terminated with semi colon.

When loop starts it executes all statement in block once and then tests the condition, after "while" keyword. If it is true it enters the loop again otherwise transfers control to the next statement after loop. It is also indefinite repetition loop.

Fig: 4.6:
while Loop Statement's Flowchart

Syntax:

```
do
{
   Statement 1;
   Statement 2;
   Statement 3;
.
.

}
while( condition );
```

Following program takes salaries of employees as input in a loop. We press y to takes more salaries as input, any other character to end. Finally it shows total salary paid to all employees.

```
/* Program 8. "do-while" loop example */
// calculates total salary paid to all employees
#include<stdio.h>
#include <constream.h>
using namespace std;
int main(void)
{
      long count=1;
      float tot_sal=0,salary;
      char ch;

      clrscr();
      do
      {
      cout<<"\n Enter salary of employee e"<<count<<"->";
      cin>>salary;
      tot_sal=tot_sal+salary;
      cout<<" Press Y to enter more salaries ";
      ch=getche();
      count++;
      }
      while(ch=='y');

      cout<<"\n Total salary paid is "<<tot_sal;
return 0;
}
```

Nested loops

You can use one or more loop inside any another 'for', 'while' or 'do while' loop. If a loop exists in the body of another loop then it is called nested loop. The inner nested loop is completely executed every time for each repetition of outer loop.

The following program is of nested for loop. It prints numbers from 1 to 10 in five rows. The outer loop shows row number, then inner loop prints numbers from 1 to 10 in one row then outer loop changes the row using '\n'. The outer loop runs five times, so inner loop which prints 1 to 10 numbers will run five times and prints numbers in five rows.

```
/* Program 9.   nested loops example */
// prints numbers from 1 to 10 in five rows
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
      int i,j;

      clrscr();
      for(i=1;i<=5;i++)
            {
            cout <<" Row no. "<<i<<" -> ";
            for(j=1;j<=10;j++)
                  cout<<j<<"  " ;
            cout<<"\n";
            }
return 0;
}
```

4.3.4 Difference between for, while and do-while loop structures

"for" Loop	"while" Loop	"do while" Loop
"for" loop is usually used in situations where we know at the start of loop that how many times loop block will execute. It is also called definite repetition loop.	It is usually used in situations where we do not know at the start of loop that how many times loop block will execute. It is also called indefinite repetition loop.	It is usually used in situations where we do not know at the start of loop that how many times loop block will execute. It is also called indefinite repetition loop.
It is called counter controlled loop as loop is controlled by a counter value, at each iteration counter value will increase or decrease	It does not need a counter value for its execution.	It does not need a counter value for its execution.
It has three parts first initialization, second condition testing and third increment or decrement.	It has only one part which is condition testing. If needed initialization is done before loop and increment or decrement is done in loop body.	It has only one part which is condition testing. If needed initialization is done before loop and increment or decrement is done in loop body.
It is pre-test loop as condition is tested at start of loop.	It is pre-test loop as condition is tested at start of loop.	It is post-test loop as condition is tested at the end of loop. So this loop executes at least once.

Teachers Note	Teacher are supposed to show step by step output of nested loops.

Recognize the use of jump statements:
- Break Statement
- Continue Statement
- Goto Statement
- Return Statement

4.4 JUMP STATEMENTS

Jump statements change execution of program from its normal sequence.

Following are the jump statements used in C++

1. break
2. continue
3. goto
4. return
5. exit ()

1. "break" Statement:

It terminates the loop or switch statement and transfers control to the statement immediately following the loop or switch statement.

e.g.

```
/* Program 10.   'break'  statement example */
// Adds five numbers if 0 is given ends program
#include<stdio.h>
#include <constream.h>
using namespace std;
int main(void)
{
int  i, num, sum;
i=0; sum=0;

cout<<"\n Enter five numbers to add. Enter 0 to terminate -> ";
      while(i<5)
      {
            cin>>num;
            if(num==0)
            {
                  cout<<"\n Ending program ";
                  break;
            }
```

```
            sum=sum+num;
            i++;
      }
cout<<"\n sum of  " <<i<< " number(s) is " <<sum;
return 0;
}
```

2. "continue" Statement:

It causes the loop to skip the remaining statements of its body and immediately transfers control to the top of the loop i.e. first statement.

e.g.

```
/* Program 11.   'continue' statement example */
// Adds five positive numbers. It does not take -ve numbers
#include<stdio.h>
#include <iostream.h>
using namespace std;
int main(void)
{
int i, num, sum;
i=0; sum=0;
  cout<<"\n Enter five positive numbers-> ";
      while(i<5)
      {
            cin>>num;
            if(num<=0)
            {
                  cout<<"\n Enter positive number";
                  continue;
            }
            sum=sum+num;
            i++;
      }
cout<<"\n sum of  five positive numbers is " <<sum;
return 0;
}
```

3."goto" Statement :

A "goto" statement jumps or transfers control unconditionally from the "goto" to a labeled statement in the same function. A labeled statement is any identifier followed by a colon (:). It is not advised to use "goto" statements in programs.

e.g.

```
If ( marks<20)
          goto warning;   // warning is label
            ….
warning: cout<<" Need very hard work in examination";
```

Control of program may be transferred to another position in program by two other ways; using return statement or exit () function.

4. "return" statement :

It terminates the execution of a function and transfers program control to the statement just after the function call statement in the calling function (or to the operating system if you transfer control from the main function). It can also return a value from the current function, if return type is not "void".

Syntax : return [expression/value];

The value of the expression clause is returned to the calling function.

5 ."exit ( )" Function:

The exit() is used to terminate a C++ program and closes program whenever executed. It is defined under "stdlib.h" header file.

Syntax: void exit (int);

- C++ has three types of control statements: Selection/Decision Making Structure, Iteration / Loops and Jump.
- C++ has three decision making structures; 'if' statement, 'if-else' statement and 'switch' statement.
- If statement checks a condition, if it is true the statements in if block are executes and if it is false, it leaves the statements in if block and starts executing statements after the block.
- If else checks a condition, if it is true the statements in if block are executed and in case it is false, it executes statements in 'else' block.
- Switch statement checks different constants after case statement with switch variable; if matches it executes statements after it otherwise goes to default statement if present.
- Loops allow us to execute a statement or a group of statements several numbers of times.
- **"for"** loop execute a sequence of statements multiple times. And is usually used in situations where we know at the start of loop that how many times loop body will execute. Condition is tested at the start of loop.
- Like for loop **"while"** loop also repeats a statement or group of statements several numbers of times while a given condition is true. It tests the condition at start of loop and is usually used in situations where we do not know at the start of loop that how many times loop block will execute.
- **"do while"** loop is similar to **"while"** loop, except that it tests the condition at the end of the loop body. So its statements block is executed at least one time.
- If a loop exists in the body of another loop then it is called nested loop.
- A **"break"** statement terminates the loop or switch statement and transfers control to the statement immediately following the loop or switch statement.

- A **"continue"** statement causes the loop to skip the remaining statements of its body and immediately transfers control to the top of the loop.
- A **"goto"** statement jumps or transfers control unconditionally from the **"goto"** to a labeled statement in the same function.
- A **"return"** statement terminates the execution of a function and transfers program control to the statement just after the function call statement in the calling function.
- The **exit()** is used to terminate a C++ program.

1. Encircle the correct answer:

i) Loop within a loop is called _________ loop.

a. inner b. outer
c. enclosed d. nested

ii) "case" and _____ are also part of "switch" statement.

a. have b. default
c. for d. if

iii) "for" Loop expression has _______ parts.

a. one b. two
c. three d. four

iv) exit( ) function is used to __________.

a. close function b. close loop
c. close program d. close switch

v) "continue" statement takes control to the _________ .

a. top of loop b. end of loop
c. top of function d. end of function

vi) In "goto" statement label is followed by ______ character.

a. colon (:) b. semi colon (;)
c. single quote (') d. double quote (")

vii) To send value to the calling function we use __________ statement.

a. throw b. return
c. send d. back

viii) "break" statement is used with ________.

a. if
b. switch
c. for
d. while

ix) Using "else" is ______ with "if" statement.

a. prohibited
b. advised
c. compulsory
d. optional

x) "if" and loop expressions use ________ operators to test condition.

a. arithmetic
b. relational
c. insertion
d. bitwise

2. Answer the following questions:

1. What is the purpose of "**default**" statement in C++?
2. Can we use "**while**" loop in place of "**for**" loop, if yes then how?
3. What is the main difference between **while** and **do while** loops?
4. Write the function of **for** loop.
5. Why we make block of statements using braces?
6. Which data type variables can be used in "**switch**" statement?
7. What is the purpose of jump statements?
8. Write the purpose of following statements:
   a. else if
   b. switch
   c. goto
   d. exit

3. Match the column:

S.no	A	S.no	B
(a)	if	(i)	Relational operators
(b)	loop	(ii)	break
(c)	Conditional expression	(iii)	switch
(d)	Loops and switch	(iv)	operator
(e)	do	(v)	iteration
(f)	>>	(vi)	else
(g)	case	(vii)	while

LAB ACTIVITIES

1. Write a program that takes a number as input and print whether it is odd or even.
2. Write a program to add numbers from 1 to 20.
3. Write a program that take month number as input (from 1 to 12) and print number of days in that month. If wrong number is given then show error message.
4. Input a number up to six digits and show each digit in separate line.
5. Take input a character, number of rows and number of columns. Draw a square box filled with that character with given number of rows and columns.
6. Write a programs that generate the following outputs

*	1	1
**	12	22
***	123	333
****	1234	4444
*****	12345	55555

7. Write a program that takes a number as input and print whether it is prime or not.
8. Take salary as input and on its basis show different levels of designations in an organization like manager, supervisor, worker etc.
9. Write a program that prints square of numbers from 1 to 10.
10. Take a number and print its table from 1 to 10 using while loop according to the following format.

$2 \times 1 = 2$
$2 \times 2 = 4$
$2 \times 3 = 6$
.
.
.
$2 \times 10 = 20$

Teachers Note	Teachers are supposed to encourage students to develop different programs by using the concepts of input/output statements, loops and selection statements.

return type

function name

parameters (arguments)

HEADER

```
int heading ( void )
{
    //statements
    return 0;
}
```

NO semicolon

BODY

- Define the term function.
  - Function Declaration.
  - Function Definition
  - Function Call
- Differentiate between function call and function definition

5.1 INTORDUCTION TO FUNCTIONS

A set of statements written to perform a specific task and having a unique name is called a function. In structured programming, the complicated and large program coding is broken down into smaller modules which are called subprograms. In C++, subprograms are called functions. Every program has at least one `main()` function in C++. When the program starts, the `main()` function is called for execution.

There are two types of functions.

1. Pre-defined Functions
2. User-defined Functions

5.1.1 Pre-define Functions:

The pre-define functions are the part of every high-level programming language. It can be used for different purposes. Predefined functions are also known as System-defined or library functions.

These functions do not need to be declared and defined. Pre-define functions are declared in header file. All predefined functions can be used simply by calling the function like sqrt(), strcpy(), toupper(), pow() etc. Many pre-define functions need proper header file by using #include pre-processor directive. The definitions of many common functions are found in the cmath and cstdlib libraries.

Teachers Note	Teacher should explain briefly Pre-define function and give some examples of pre-define function for the practice of students.

5.1.2 User – defined-function

Programmer can also write their own functions to perform specific task. They are called user-define-functions. These functions need declaration and definition. When the user-defined function is called from any part of the program, it will execute the code defined inside the body of the function. multiply ( ), sum( ), average( ) may be the example of User–define functions. A user-define function based on two parts:

1. Function declaration or prototype
2. Function definition

1. Function Declaration:

A function without its definition (code block) is known as a function declaration or function prototype. It is declared before the main( ) function. A function declaration tells the compiler about the function's name, return data types, and arguments/parameter data types and it ends with statement terminator (;).

Syntax:

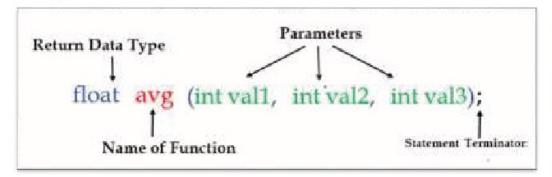



Fig. 5.1 Function Declaration

a. Return Data Type:

It shows the data type of value returned by function. It may be int, float, double and char data type. If no value is returned by the function in that case keyword "void" is used.

b. Function Name:

It specifies the name of function. It is recommended that meaningful and understandable names are given to the function.

c. Parameters:

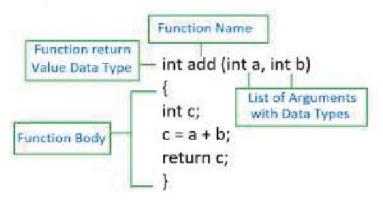
It define the list of data types of function parameters that are to be passed to the function. Parameters are separated by commas. Parameters are also known as arguments. If there is no parameter in function, programmer uses keyword "void". Variable names are optional in prototype parameters/ arguments.

d. Statement Terminator:

In function declaration, statement terminator must be used at the end.

2. Function Definition:

Function definition is function itself. It has a function header and a body or code block. Header has three parts; return value data type, function name and list of augments with datatypes in parenthesis. Body of the function includes statements in braces. Function definition may be defined before or after the main function.

For example:

Function call

To use the function code, we have to call or invoke that function with its name. It is called function call. When function is called for execution, control will transfer to the function definition and all statements of function definition will execute and after executing the statements the control will transfer back to the calling function (statement after function call).

If the function is without return value and no arguments then it is called by its name. It means function's braces will be empty.

Syntax: **function_name ( );**

Example: add( );

If function returns a value, then we can store return value to a variable in the calling function.

Example: x=add (y, z);

Function passing argument or parameters:

An argument is a part of data passed to the function. When function is called for execution, the actual values as parameters are also given with function call statement. Passing actual values to function as arguments with function call statement are known as actual parameters. Actual parameters may be variables or constants. They are placed in paratheses after the function name. These values are received in variables of the header of function definition. These receiving variables are called the formal parameters of the function. It acts as a local variable inside the function in which they are used.

Returning value from Function:

In C++, the return keyword allows a function to return a value. When a function completes its execution, it can return a single value to calling function. Return data type must be specified with the function header in the function definition as well as function declaration. It is written before function name.

Syntax: int function name( );

Differentiate between function definition and function call

Function definition and function call can be differentiated on the basis of following criteria.

<table>
<tr><th>Function definition</th><th>Function call</th></tr>
<tr><td>The function definition is function itself. It has header and body with statements. Function definition may appears before or after the main ( )function.</td><td>Function call is to invoke the code of function by its name. As the function is called, control is transferred to the called function.</td></tr>
<tr><td>Syntax:
<code>data_type function_name</code>
<code>(parameters list)</code>
<code>{</code>
<code>statements;</code>
<code>}</code>
Example
int sum(int p, int q)
{int z;
z= p + q;
return z;
}</td><td>Syntax:
<code>variable_name= function_name</code>
<code>(parameter list);</code>
Example
a=sum (x, y);</td></tr>
</table>

Different Ways to Use User-Define Function: based on argument / parameters and return type

There are four different methods in C++ based on passing parameters to the function and return values from the functions.

- No return value and No passing arguments/parameters

  void function name(void);

- Return value but No passing arguments/parameters

  int/float/char function name**(void);**

- No return value but Passing arguments/parameters

  void function name(int, float, char);

- Return value and Passing arguments/parameters

  int/float/char function name**(int, float, char);**

Teachers Note	Teacher should explain how to write a User-define function in four different methods in C++.

Using the Pre – defined Function in C++ Program

```
//Using Square Root formula in program
#include<iostream.h>
#include<cmath.h>
using namespace std;
int main(void)
{

   int a;
   cout << "\n Enter the value of a........";
   cin >> a;
   cout << "\n The square root of a is......" << sqrt(a); //
Pre-defined function
   return 0;
}
```

OUTPUT

Enter the value of a........4
The square root of a is....16

Applying the User-defined function in the program.

```
/* Program of an average of two numbers by using User-define
function */
#include<iostream.h>
using namespace std;

flaot average(int x, int y);
int main(void)
{
    int x,y;
    average(x,y);

return 0;

}
```

```
float average(int x, int y)
{
  cout << "\n \t Enter the value of x.........:";
  cin  >> x;

  cout << "\n \t Enter the value of y.........:";
  cin  >> y;

  avg = (x+y) / 2;

  cout << "\n The average of two numbers is.....” << avg;
  return avg;

}
```

OUTPUT
Enter the value of x……..24
Enter the value of y……..34
The average of two number is…..29

Differentiate between Pre-define and User-define Function.

Pre – define function/System Define	User – define function
These are the library functions.	These functions are created by the programmer.
It cannot be modified.	It can be modified by the programmer.
No need for function definition as that is part of C++ compiler.	Their declaration and definition are needed in the program.
Example: gets(), putchar(), getch(), sqrt(), etc.	Example: add(), int sum(); float avg(float a, float b) etc.

Local variable and global variable

In structured programming, generally two types of variables are used.

1. Local Variables
2. Global Variables

1. Local Variables:

They are declared inside any function. A local variable is only accessible within a specific part of a program.

For Example:

```
#include<iostream.h>
using namespace std;
int main(void)
{
int a=10;  // Local Variable
cout << "\n The value of a....." << a;
return 0;
}
```

Teachers Note	The variables declared in the header of the function definition are also known as Local variable.

2. Global Variable:

Global variables are declared outside of the main function. It is also known as **external variable.** It holds the value of variable during the entire execution of the program. The value of global variables can be shared with different functions.

For Example:

```
#include<iostream.h>
using namespace std;
void add(int c);
int a=20;  // Global Variable
int main(void)
{
      int x;    //Local Variable
      cin>>x;
      add(x);
      cout << "\n the value of a is......" << a;

 return 0;
 }

      void add (int c)
      {
            int b;   //Local Variable
            b=a+c;
            cout << "\n the addition of two numbers is....."
      << b;
      }
```

Here 'a' is global variable and 'x' and 'b' are local variables. Receiving arguments are also local variable as 'c' variable in this program.

Teachers Note	• Teacher are supposed to demonstrate practically about difference between local and global variables in C++. • At this point students should be able identify and remove errors from programs.

- A group of statements written to perform specific task is called Function.
- Function in C++ helps the programmer to manage the code of the large program.
- Functions are divided into three sections:
    1. Function declaration
    2. Function definition
    3. Function calling
- Function declaration tells the compiler about the function name, return types and parameters types.
- Function call is to invoke the code of function by its name.
- Functions are divided into two categories.
    - User – defined-function
    - Pre –defined – function
- A programmer can write his/her own function which is called User – defined function.
- In C++, Pre –defined – function are already declared in header files.

A. ENCIRCLE THE CORRECT ANSWER:

1. The functions that are defined by the programmer are called:
   a. Built-In function b. User-defined-function
   c. Subfunction d. Function
2. A programmer creates a function for a particular task and the programmer wants to include that function in program. Which extension is required to save that function?
   a. .obj b. .h
   c. .cpp d. .exe
3. In C++, int main( ) returns which data type value by default?
   a. float b. int
   c. char d. double
4. The parameters specified is the function header are called:
   a. formal parameters b. actual parameters
   c. default parameters d. command line parameters
5. The word "prototype" means:
   a. Declaration b. Calling
   c. Definition d. Both a & b
6. The function prototype consists of:
   a. Name of function
   b. the parameters are passed to the function
   c. The value return from function
   d. All of these
7. All variables declared in function definition are called
   a. Local variable b. Instance variable
   c. Global variable d. Static variable
8. Which are not the built-In function?
   a. sqrt() b. time()
   c. exp() d. sin()

B. RESPOND THE FOLLOWING:

1. Differentiate between function declaration and function definition.
2. What is the purpose of keyword "void" in function?
3. Why we use header files?
4. Differentiate between passing argument and return the value from function.
5. What is the difference between external variables and function local variables?
6. List the five standard built-In functions with examples.
7. Write down the advantages of User – define functions in C++.
8. Get the output and highlight the errors from the following program.

```
#include<iostream>
void Table(void);
void Table(void)
{
      int m,n;
      cout << "\n The value of m........";
      cin >> m;
      for(n=1; n<=10; ++n)
{
      cout <<"\t "<<m<<"*"<<n<<"="<<m*n<<"\n";
}
}
      void main(void)
{
      Table();
}
```

LAB ACTIVITIES

1. Write a program on the following given series by using For loop.

   0, 5, 10, 15, 20, 25, 30, 35

   Apply the technique (no return value and no pass parameters) in program.

2. Write a program to take input from the keyboard and check whether given number is Even or Odd. Apply the technique (return value and pass parameters) in program
3. Write a program to convert kilogram in grams using function. The function should take value in kilogram as parameter and should return value in grams.
4. Create a function that takes length and height as arguments and print a box of stars accordingly.

   e.g. length =10, height = 4

   ****

   ****

   ****

   ****

   Apply the technique (return value but not pass parameters) in program.
5. Write a function that returns factorial of a given number.
6. Develop programs for manipulating the following formulas in form of function.

Title	Formula	Description
Area of Rectangle	a= l b	area = length x width
Area of Circle	$a = \pi * r^2$	area = pi x radius x radius (pi = 3.14)
Pythagorean Theorem	$c^2 = a^2 + b^2$	*****

DIGITAL LOGIC AND DESIGN

- Recall that data is represented using binary pulses (0 and 1)
- Explain that binary pulses have a respective low and high voltage
- Explain the three basic logic gates.
- Construct the 2, 3, 4…n variable truth tables for basic logic gates.
- Explain the universal gates with the help of truth tables.
- Differentiate between basic and universal logic gates.

6.1 DATA REPRESENTATION IN A COMPUTER

Data Representation refers to the form in which data is stored, processed, and transmitted. Digital devices such as smartphones, iPads, and computers store data in digital formats that can be handled by electronic circuitry. These circuits work on two logical binary states i.e., Low and High (1 & 0) pulses.

Fig. 6.1 Digital Signal

Logic gates are the electronic circuits in a digital system. Logical gates perform logical operations like AND, OR, NOT, NAND, NOR etc. are shown in Figure No. 6.1

The Binary logic consists of binary variables and logical operations. The variables are denoted by letters of alphabets such as A, B, C,… ,Z, a,b,c,…z, e.g., with each variable having only two distinct possible logical values 1 and 0, these two values of variables may also be identified by different names (e.g., true and false, Yes or No).

6.2 LOGIC GATES

Logic gates are the electronic circuits in a digital system. Logical gates perform logical operations like AND, OR, NOT, NAND, NOR etc. Logic gates are divided into two categories.

6.2.1 Basic Logic Gates

6.2.2 Universal Logic Gates

6.2.1 Basic Logic Gates:

The logic gate is the basic unit of digital logic circuits, there are mainly three basic gates AND, OR, and NOT and these logical gates perform AND, OR, and NOT operations in the digital system.

A truth table is a tabular representation of all the combinations of values for inputs and their corresponding outputs.

AND GATE:

An AND gate is a digital circuit that has two or more inputs and a single output. AND gate operates on logical multiplication rules. AND operation using variables A and B is represented "A.B", here (.) dot is a logical multiplication sign.

Boolean Expression of AND gate: Y=A.B

Truth table of and operation using two input variables

A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

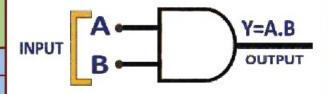



Fig. 6.2
AND Gate using two input variables

Truth Table of AND gate using two input variables A, B and output is Y. If any input is 0, then output Y becomes 0. If all inputs are 1 then output Y becomes 1.

Truth table of and operation using three input variables			
A	B	C	Y = A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

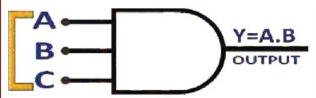
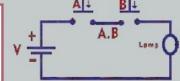



Fig. 6.3
AND Gate using three input variables

AND gate using three input variables A, B, C, and output is Y. If any input is 0, then output Y becomes 0. If all inputs are 1 then output Y becomes 1.

Teachers Note

Teacher gives the concept of implementation of AND operation for the logical input and output using given diagram.

A,B open for logic 0
A,B closed for logic 1
When Lamp is OFF for A.B=0 (Logical)
When Lamp is ON for A.B=1(Logical)

OR GATE:

An OR gate is a digital circuit that has two or more inputs and a single output. OR gate operates on logical Addition rules. Logical OR operation using variables A and B is represented as "A+B", here (+) is a logical Addition sign. Boolean Expression of OR gate is Y=A+B.

Truth table of or gate operation using two input variables		
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

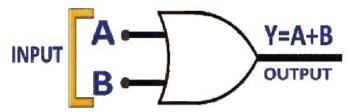



Fig. 6.4 OR Gate using two input variables

Truth Table of OR gate using two input variables A, B and Y is output. If any input is 1 then output Y becomes 1 and if all inputs are 0 then ouput Y becomes 0. Boolean expression of OR gate is Y=A+B.

Truth table of or operation using three input variables			
A	B	C	Y = A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

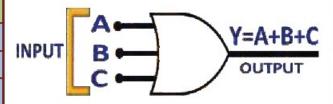
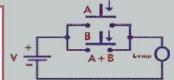



Fig. 6.5
OR Gate using three input variables

Truth Table of OR gate using three input variables A, B, C and Y is output. If any input is 1 then output Y becomes 1 and if all inputs are 0 then ouput Y becomes 0.

Teachers Note	Teacher gives the concept of implementation of AND operation for the logical input and output using given diagram. A,B open for logic 0 A,B closed for logic 1 When Lamp is OFF for A+B=0 (Logical) When Lamp is ON for A+B=1 (Logical)

NOT GATE:

A NOT gate is a digital circuit that has a single input and a single output. It is also known as an INVERTER. The output of NOT gate is the logical inversion of input. It is symbolically represented by complement sign (′) Right side on top of the input variable or bar(-) sign on top of the variable. Boolean expression of NOT gate is $Y = A'$ or $Y = \overline{A}$

Truth table of not gate operation using two input variables	
A	$Y = \overline{A}$
0	1
1	0

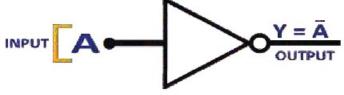



Fig. 6.6 NOT Gate

Truth table of NOT gate is A as input and $Y = \overline{A}$ is output.

6.2.2 Universal Gates:

A **universal gate** is a logic gate which can implement any Boolean function without the need to use any other type of basic gates. The NOR gate and NAND gate are universal gates.

NAND(NOT – AND) GATE:

The NAND gate or "Not AND" gate is the collection of two basic logic gates, the AND gate and the NOT gate connected in series.

Boolean expression of NAND gate is $Y= (A.B)'$ or $Y = \overline{AB}$.

Truth table of nand operation using two input variables		
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

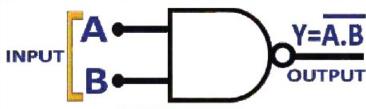



Fig. 6.8 NAND Gate

TheTruth table of the NAND gate using two input variables A, B and Y is the output. When all inputs are "1", the output, Y is "0". If any one of the inputs is "0", then the output Y is "1".

NOR GATE:

A NOR Gate is the collection of OR Gate NOT Gate. The output of NOR gate is inverter OR. The Boolean expression of NOR gate is Y = (A+B)′ or $Y = \overline{A+B}$.

Truth table of nor operation using two input variables		
A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

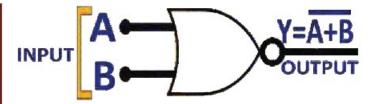



Fig. 6.9 NOR Gate

The Truth table of the NOR gate using two input variables A, B and Y is the output. If both inputs are "0", then the output, Y is "1". If any one of the inputs is "1", then the output Y is "0".

Teachers Note	Teacher will demonstrate thoroughly the concept and operations of NAND & NOR universal gates. And also compare the symbols of NOT gate and Inverter gate.

Differentiate between basic & universal logic gates

Basic logic gates	Universal logic gates
AND, OR, and NOT gates are the most basic logic gates. By using this set of logic gates, it is possible to implement all the possible Boolean Expressions by using these three gates.	The NAND gate and NOR gate can be called the universal gates, the collection of NAND & NOR gates can be used to achieve any of the basic AND, OR and NOT operations.
Individual logic gates can be connected to form a variety of different combinational logic circuits.	A universal gate is a gate which can implement any Boolean expression without using other type of gate.

- Recall that data is represented using binary pulses (0 and 1)
- Apply 12 rules of Boolean algebra for simplification of any expression
- Design a logic circuit for any Boolean expression
- Derive Boolean expression for any logic circuit

6.3 BOOLEAN ALGEBRA

Boolean algebra was invented by George Boole in 1954. It is a branch of mathematics and it can be used to describe the logical operations and processing binary information. It is based on true or false input values to produce a true or false output value.

6.3.1 Rules of Boolean Algebra:

The Boolean arithmetic rules are pre-defined rules that help to simplify the logical expression. There are 12 basic rules which are invented to simplify the gates. To reduce the number of logic gates needed to perform a particular logic operation we can apply a set of rules. These rules are commonly known as the Laws of Boolean Algebra Expressions.

The following table shows some of the Boolean algebra rules for Boolean Expression Simplification.

Rule No.	Boolean Expression Simplification Rules	Prove these rules with Proof according to their Values
1.	**A + 0 = A**	If A=0 ⟶ 0 +0=0 If A=1 ⟶ 1+0=1
2.	**A + 1 = 1**	If A=0 ⟶ 0+1=1 If A=1 ⟶ 1+1=1
3.	**A . 0 = 0**	If A=0 ⟶ 0.0=0 If A=1 ⟶ 1.0=1
4.	**A . 1 = A**	If A=0 ⟶ 0.1=0 If A=1 ⟶ 1.1=1
5.	**A + A = A**	If A=0 ⟶ 0+0=0 If A=1 ⟶ 1+1=1
6.	**$A + \overline{A} = 1$**	If A=0 ⟶ $0+(\overline{0})$ ⟶ 0+1=1 If A=1 ⟶ $1+(\overline{1})$ ⟶ **1+0=1**
7.	**A . A = A**	If A=0 ⟶ 0.0 =0 If A=1 ⟶ 1.1 = 1
8.	**$A . \overline{A} = 0$**	If A=0 ⟶ $0.(\overline{0})$ ⟶ 0.1=0 If A=1 ⟶ $1.(\overline{1})$ ⟶ 1.0=0
9.	**$\overline{\overline{A}} = A$**	If A=0 ⟶ $(\overline{0}) = 1$ ⟶ $(\overline{1}) = 0$ so, $\overline{\overline{A}} = 0$ If A=1 ⟶ $(\overline{1}) = 0$ ⟶ $(\overline{0}) = 1$ so, $\overline{\overline{A}} = 1$
10.	**A + A . B = A**	If A=0, B=0 ⟶ 0+(0.0) ⟶ 0+0 = 0 If A=1, B=1 ⟶ 1+(1.1) ⟶ 1+1 = 1
11.	**$A + \overline{A} . B = A + B$**	If A & B=0 ⟶ $0+\overline{0}.0$=0+0 ⟶ 0+**1**.0=0+0 ⟶ 0+0 = 0+0 Hence **0=0** If A & B=1 ⟶ $1+\overline{1}.1$=1+1 ⟶ 1+**0**.1 =1+1 ⟶ 1+0=1+1 Hence **1=1**

12.	(A+B) (A+C)= A + BC	If A,B& C=1 ⟶ (1+1)(1+1)=1+(1.1) ⟶ **(1)(1) = 1+(1)** ⟶ **1.1=1+1** ⟶ **1=1** If A, B=0 and C=1 ⟶ (0+0) (0+1)=0+0.1 ⟶ (0).(1)=0+0 ⟶ 0.1=0 ⟶ 0=0 Hence **0 = 0**

6.3.2.1 Example to apply Boolean Rules for Simplification expression.

a. $AB+A\overline{B}=A$

Solution:

$AB+A\overline{B}=A$

Take L.H.S

Here take common variable A

$A(B+\overline{B})$

According to 6th Rule of Boolean Algebra is $A+\overline{A}=1$ so, $B+\overline{B}=1$

A. (1) Or A.1 ⟶ According to 4th Rule of Boolean Algebra i.e.A.1= A

A .1=A

A=A

Hence **L.H.S = R.H.S**

A =A**Proved**

b. $(A+B)+(A+\overline{B})=A$

Solution:

$(A+B)+(A+\overline{B})=A$

Take L.H.S

$(A+B)+(A+\overline{B})$

ANDing (Multiplication) of both expressions

$AA+A\overline{B} + BA+B\overline{B}$

Apply Rule 7[th] of Boolean Algebra i.e. **A.A=A** and Rule 8[th] of Boolean Algebra i.e. **$A.\overline{A}$=0 0r** $B\overline{B}$

$A+A\overline{B} + BA + 0$

Take common variable A from $A+A\overline{B} +BA$ expression

$A+A(B+\overline{B})+0$

Apply Rule 6[th] of Boolean Algebra i.e. **$A+\overline{A}$ = 1 or** $B+\overline{B}=1$

$A+A(1)+0 = A+A+0$

Apply Rule 5[th]of Boolean Algebra i.e. **A+A = A**

$A+0 = A$

Hence **L.H.S = R.H.S**

A = A **Proved**

6.3.2.2 Draw Logic Circuit of the given Boolean expressions.

a. $Y = \bar{A}BC\,\overline{(A + D)}$

Solution:

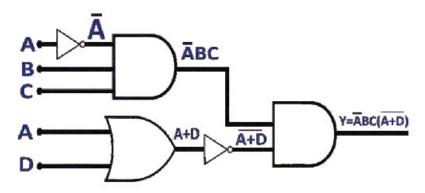



Fig. 6.10 Logic Circuit

Explanation:

Above logic circuit consists of AND, OR and, NOR gates. The expression $\bar{A}$ NOT and B, C gate connected with AND gate and A, D is a connected with OR gate and converted in NOR gate. Finally, $\bar{A}BC$ and $\overline{A + D}$ connected to **AND** gate.

b. $X=\overline{AB(\overline{C + D})}$

Solution:

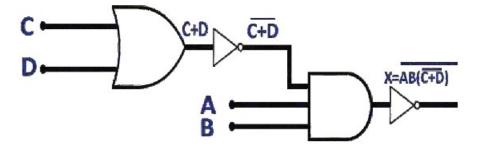
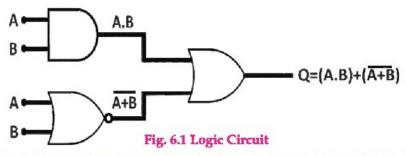
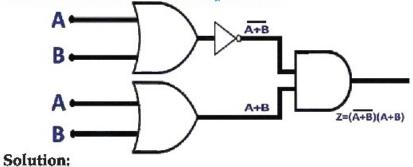



Fig 6.11 Logic Circuit

c. Let:- $Q=(A.B)+(\overline{A+B})$

Fig. 6.1 Logic Circuit

6.3.3.3 Derive the Boolean expression from the given circuit and make a truth table of that Boolean expression.

Solution:

Boolean Expression of the above circuit is $Z = (\overline{A+B})$ (A+B)

Truth table of the $Z = (\overline{A+B})$ (A+B)

INPUTS		TRUTH TABLE		
A	B	A+B	$\overline{A+B}$	$(\overline{A+B})$.(A+B)
0	0	0	1	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	0

Teachers Note	Teacher usually find it difficult to teach Digital Logic & Boolean Algebra since these are abstract concepts. These concepts may be presented to students with the help of images and videos. If students can visualize these concepts, they can better assimilate them.

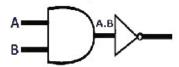
- Data Representation refers to the form in which data is stored, processed, and transmitted. Digital devices such as smart phones, iPods, and computers store data in digital formats that can be handled by electronic circuitry.
- Logic Gates are the electronic circuits in a digital system.
- Logical Gates perform logical operations like AND, OR, NOT, NAND, NOR etc.
- The logic gate is the basic unit of digital logic circuits, there are mainly three basic gates AND, OR, and NOT and these logical gates perform AND, OR, and NOT operations in the digital system.
- An AND gate is a digital circuit that has two or more inputs and a single output AND gate operates on logical multiplication rules. Boolean Expression of AND gate: Y=A.B
- An OR gate is a digital circuit that has two or more inputs and a single output. OR gate operates on logical Addition rules.
- Boolean expression of OR gate is Y=A+B.
- A NOT gate is a digital circuit that has a single input and a single output. It is also known as INVERTER.
- Universal Gates are logic gates. They are capable of implementing any Boolean function without requiring any other type of gate. There are two types of universal gates
- A NAND Gate could be construct by connecting a NOT Gate at the Output terminal of the AND Gate. Boolean expression of NAND gate is Y= (A.B)′or Y = $\overline{AB}$.
- A NOR Gate could construct by connecting a NOT Gate at the output terminal of
- The Boolean expression of NOR gate is Y = (A+B)′ or Y = $\overline{A+B}$.
- The Boolean arithmetic rules are pre-defined rules that help to simplify the logical expression. There are 12 Boolean algebra rules.

A. ENCIRCLE THE CORRECT ANSWER:

1. The universal gate is ____________________.
   a. NAND Gate b. AND Gate
   c. OR Gate d. None of these
2. The __________ is Inverter.
   a. AND b. OR Gate
   c. NOT d. None of these
3. In Boolean Algebra, the bar sign (-) indicates __________________.
   a. OR Operation b. NOR Operation
   c. NOT Operation d. Both b and c
4. The Boolean Algebra is used for ____________________.
   a. Creating Circuits b. Apply 12 rules of Boolean
   c. Simplify the Boolean expression d. Differentiate the gates
5. With the combination of three variables, how many outputs are expected altogether?
   a. Three b. Six
   c. Eight d. Nine
6. A+A=A is a __________________ rule of Boolean Algebra.
   a. 3rd b. 6th
   c. 5th d. 7th
7. $A.\overline{A}=0$ is a ___________________ rule of Boolean Algebra.
   a. 1st b. 8th
   c. 6th d. 10th
8. Simply form of Boolean expression of $ABC + AB\overline{C} + \overline{A}B$ is _______________.
   a. A b. B
   c. C d. $\overline{B}$

9. Simplify form of Boolean expression of $ABC+\overline{A}BC$ is ______________.

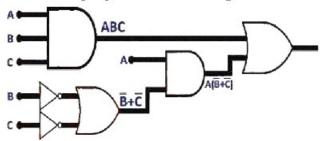
a. AC
b. BC
c. B
d. $\overline{A}$

10. Boolean expression of the given circuit is ________________________.

a. $\overline{A + B}$
b. B.A
c. A.B
d. $\overline{A . B}$

B. RESPOND THE FOLLOWING:

1. Explain all basic logic gates and their operations.
2. Differentiate between NAND and NOR gates.
3. Why do we use Boolean Algebra?
4. Explain the function of Inverter.
5. Explain the purpose of Truth table
6. Simplify the following Boolean expression.
   $Z = AB + A(B + C) + B(B + C)$

LAB ACTIVITIES

1. Draw a Logic Circuit of 10^{th} Law of Boolean Algebra.
2. Design a Logic Circuit from Boolean expression $Q = (A.B) + (A.B) + (\overline{A + B})$
3. Derive the Boolean expression from the given circuit and make a truth table and simplify that Boolean expression.

INTRODUCTION TO SCRATCH

SCRATCH

- Explain scope, possibilities and limitations of scratch.
- Demonstrate downloading and installation process of Scratch Editor OR Working with Scratch Online.

7.1 SCRATCH

Since the emergence of FORTAN in 1950s, computer languages have evolved enormously. Programming languages are used to create instructions for computers. Using these limited instructions, we can instruct the computer to perform a desired activity. Nowadays, programming languages have become very user-friendly and easy to learn and program. Now, there are some visual languages which do not require memorizing of code syntaxes. Instead, a program is developed by simply dragging and dropping some components of that language and entering their values.

Scratch is one of such programming languages. It is very easy to create programs in Scratch which include games and animations. Interactive stories, games, animation, music, art, and presentations can be created by simply dragging and dropping colored blocks. This programming tool was developed by Massachusetts Institute of Technology-(MIT) Media Lab. Scratch is free to use and distribute and will remain free forever.

When students create programs, they learn important mathematical and computing concepts that improve their creative thinking, logical reasoning, problem solving, and collaboration skills. Scratch can be used for multiple purposes. Kids can make animations, teens can develop game and even teachers and students can use it to create effective education tools such as math quiz, science simulation, and educational videos. Scratch is so easy that anyone can master using it within very short period.

Scratch can either be downloaded from MIT website for offline use or used online directly in the web browser.

7.1.1 Downloading and installing scratch offline

You can download and install Scratch on a computer or an android device to work offline. The latest version of scratch can be downloaded for Windows 10 or Android 6 or later versions. However, Scratch 2.0 is a good version that works on most of the computers. Scratch 2.0 can be downloaded from this website: https://scratch.mit.edu/download/scratch2

The instructions in this unit are based on Scratch 2.0. However, teachers and students can use the latest version as well.

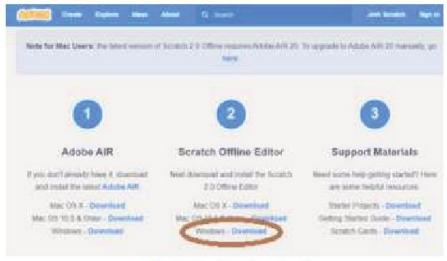



Fig.7.1 Scratch Downloading

To begin, **Scratch Offline Editor** for Windows needs to be installed on to your computer.

Installation of Scratch is very easy.

1. After downloading we need to just run the executable (exe) file.
2. The first screen will appear which will ask the location where we want to install the Scratch.
3. It will also ask about the shortcuts to create. You need to just continue with default options.
4. This will initiate the installation process and after installation, Scratch will be ready to use.

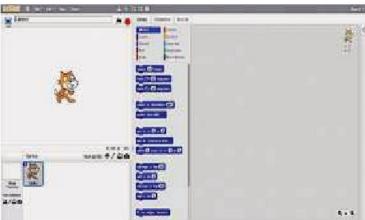



Fig.7.2 Scratch Editor

7.1.2 Scratch online

Scratch can also be used online by simply loading its editor in our web-browser which is available at:
https://scratch.mit.edu/projects/editor
After loading the editor, it will function just like the offline editor. We can also create our account on Scratch. This will enable us to save our projects online.

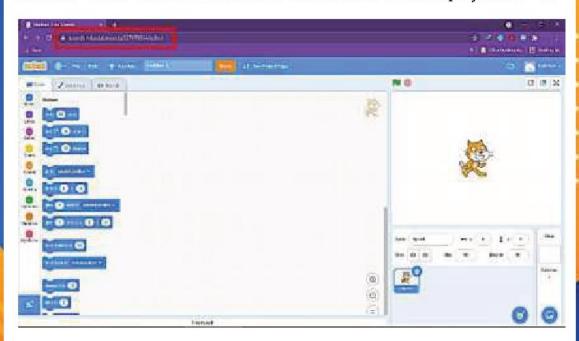



Fig.7.3 Scratch Online

- Explain the environment and tools in Scratch including sprite and scripts
- Demonstrate the use of Code, Costumes and Sound Tabs

7.2 UNDERSTANDING SCRATCH ENVIRONMENT

Before we start understanding the environment of Scratch, we must understand the two basic concepts, i.e., **Sprites** and **Scripts**.

Sprites	The sprites are the images of cartoons, characters or objects that we add in our project. We can have multiple sprites in our project but at least one sprite is always needed for the project. Cat is the default sprite in the Scratch.	
Scripts	To create a game, interactive story, animation or artwork in Scratch, you must add visual instructions to tell a sprite exactly what to do. The scripts are instructions that make sprites perform a task. Each sprite in a Scratch project has an area for scripts through which it is programmed. Clicking on a sprite's thumbnail in the sprite pane will bring up the script area of that sprite.	

7.2.1 Scratch editor:

Following are the different components of Scratch Editor.

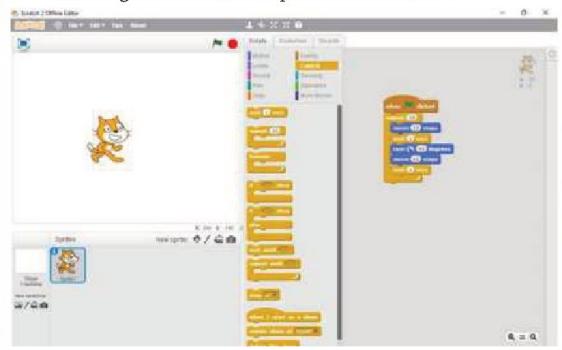



Fig. 7.4 Stage Preview

- **Stage or Stage Preview Window:**

This is where you can immediately see the output of your codes. The project runs physically in this window. It is the main working area where the sprite moves and performs actions according to the given instructions. It is divided into x (horizontal) and y (vertical) coordinates. The coordinates are displayed at the bottom right corner of the stage. These coordinates indicate the position of the sprite on the stage.

- **Script Area:**

This shows your code or program. You can drag blocks from the palette to this area and create scripts by placing them together. Script is the set of step wise instructions that you give to the Sprite to do a particular task. Each Sprite has its own Script Area.

- **Sprite List:**

It displays the thumbnails of all the sprites available in a project. You can click the blue information icon on any sprite to change its name and behavior.

- **Backdrop:**

A backdrop is the background that we can add on our stage. By default, there is no backdrop is added in a project. You may change how your stage looks by adding new backdrops.

- **Script Block:**

Script Area has three different Tabs. A Tab is a small form or area that contains similar command or options.

- **Script Tab (Code Tab in Scratch 3):**

You can think of this area as your toolbox. When you click on this Tab, the block palette will open. You tell the Sprites exactly what to do by giving them commands. A command is an instruction to do a particular task. In Scratch these commands are shown in the form of Code Blocks in the Blocks Palette. The block palette consists of every block of instruction that is built into Scratch. The commands regarding specific tasks are joined together in different blocks like Motion, Looks and Sensing. Each block has an associated color that differentiates different commands.

The following table describes the purpose of each block.

Blocks	Purpose	Examples
Motion	These codes are used to move the Sprite on the stage.	`Move; Turn; go to x,y; if on edge, bounce;`
Events	Events trigger specific code at a particular time or action.	`when flag is clicked; when space key is pressed, when backdrop switches to.`

Sound	This is used to play sound.	Play sound; play drum 1 for 0.25 beats, play note 60 for 0.5 beats.
Looks	These codes are used to change the appearance of the Sprite and Backdrop.	Say, Think, Switch Costume to and Switch Backdrop to are some commonly used codes.
Control	These codes control the actions on the stage.	wait 1 sec, repeat 10, forever, if then.
Sensing	These codes sense any specific happening	touching mouse pointer, touching color, ask and wait
Data	These are used to initialize variables and list	variable and lists
Pen	This used to draw lines, rectangles and other shapes	pen up, pen down, pen color
Operators	This shows the available arithmetic, logical and relational operators	+; -; *; /; <; >; =

- **Costume Tab:**

The appearance of a sprite can also be changed. You can change the costume of a sprite clicking on **"Costumes"** tab and clicking on the desired costume of choice, or by using blocks to select the sprite's costume. New costumes for the sprite can be Imported, Created, and Edited in the Scratch Paint Editor.

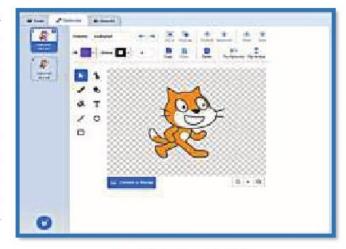



Fig. 7.5 Costumes Tab

- **Sound Tab:**

Some sprites additionally have at **least one sound**. Unlike costumes, sounds are an optional field, so you can have a sprite with no sounds. The sounds tab allows you to add, delete, and edit sounds. Sounds can be played in the sound editor or with blocks that play a specific sound.

Fig. 7.6 Sound Tab

- **Cursor Tool:**

You can find Cursor Tool on the right top of the editor. It includes five options; Duplicate, Delete, Grow, Shrink and Help. To Duplicate a sprite, just click on the stamp and then on the Sprite. Same is applicable on Delete, Grow and Shrink.

Fig. 7.7 Cursor Tab

Teachers Note	Students may have many questions at this point. Teachers are expected to spend some time to demonstrate important code with example.

- **Basics about Creating Program:**

Few points should be kept in mind before developing a project.

- Most of the Scratch Programs contain Sprites, Backdrops and Code Blocks.
- Every Sprite in a program has a separate Code Block which controls its actions.
- You can develop programs for different logics like storytelling, sprite animation, simple game and others.

Following steps are generally taken to develop a simple program in Scratch.

Open Scratch Editor

- From Events Option in Script, drag and drop on script area.
- From Control Option, drag **forever** and drop on script area. Inside the **forever** block, drag and drop the illustrated commands and change values accordingly. Colors will help you to find the option.
  After completing the codes blocks, Play (Run) and Stop the program.

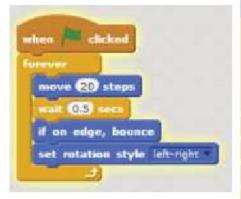



Fig. 7.8 Sample Program

- **Playing and Stopping the Animation/Program:**

To start your program or to test your code click the *Green Flag* icon located above the Stage panel. To stop your program, click the *Red Stop* icon.

What is the output of this program? Let's try to break the logic of this program.

Fig. 7.9 Play and Stop

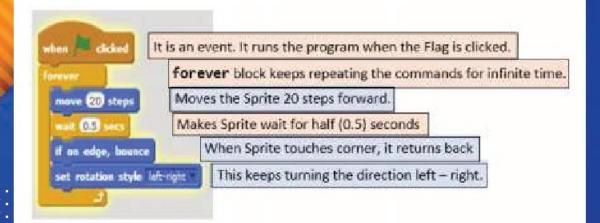



Fig. 7.10: Program Explanation

• Coordinates on Scratch Stage:

Scratch has a two-dimensional (2D) coordinate system; "X position" and "Y position" to determine the location of a sprite on the stage. The "X position" value determines the horizontal location of the sprite and the "Y position" value determines the vertical location.

As shown in figure 7.15, the screen in Scratch is a 480 × 360 rectangle. The X position can range from -240 to 240, where -240 is the leftmost a sprite can be and 240 is the rightmost, and the Y position can range from 180 to -180, where 180 is the topmost it can be and -180 is the bottommost it can be.

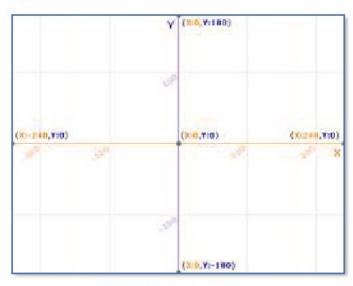



Fig 7.11 Coordinate Stage

Teachers Note	Teachers can connect the concept of the coordinates in mathematics. For better understanding some sample programs may be demonstrated and given as lab assignment.

7.2.2 Some Important Commands in Scratch:

Here are some important commands that are usually used to develop simple programs.

Command	Group	Purpose
when [flag] clicked	Event	It triggers the following code blocks when [flag] is clicked
when space key pressed	Event	It triggers the following code blocks when a specific key is pressed
move 10 steps	Motion	Moves the sprite in current direction for specified steps
turn ↻ 15 degrees	Motion	Turns Sprite to specified degree
go to x: 0 y: 0	Motion	Sends Sprite to specified x, y coordinates (position on stage)
glide 1 secs to x: 0 y: 0	Motion	Sprite glides to specified x, y coordinates (position on stage)
wait 1 secs	Control	Waits for specified seconds
repeat 10	Control	Repeats the following code blocks for specified number of times.
forever	Control	Keeps repeating the following code blocks for infinite times.
if then	Control	Executes the block when given condition is true.
play sound meow	Sound	Plays the specified sound.
play drum 1 for 0.25 beats	Sound	Plays the specified drums sound for specified beats.
play note 60 for 0.5 beats	Sound	Plays the specified note for specified beats.
set instrument to 1	Sound	Changes the musical instrument.

Block	Category	Description
say Hello! for 2 secs	Looks	Says the given words for specified seconds.
say Hello!		Says the given phrase.
think Hmm... for 2 secs		Pretends to think by showing given phrase for specified seconds.
think Hmm...		Pretends to think by showing given phrase.
switch costume to costume2		Changes the costume to the selected value.
switch backdrop to backdrop1		Changes the background to the selected value.
set size to 100 %		Changes the size of Sprite.
pen down	Pen	Pen is down. Now a line is drawn as Sprite moves.
pen up		Pen is up.
set pen color to		Sets the color of the pen to draw line.

7.3 Developing Programs in Scratch

After learning some frequently used commands, let's develop few programs. Try developing the illustrated program.

1. Changing the Costume:

With this small set of code blocks (Fig. 7.12), the costume of the Sprite changes and it looks as if the Sprite is moving.
Can you explain this program?

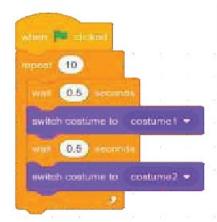



Fig. 7.12 Change Costume

2. Adding and Moving Sprite

For next program you should right click on the Sprite (Cat) and delete it. Now you don't have any Sprite in your project.

- To add a Sprite, click on "Choose sprite from library". This will open a dialogue box showing all available Sprites. Select Beetle from the list.
- On the Stage, Move the Beetle to the Upper- left corner.
- Now add the following codes.

What is the output? The beetle is moving clock-wise in rectangular shape. Can you change its movement to antilock-wise? Can you write the logic of this program in simple words?

```
when [flag] clicked
forever
  move 20 steps
  wait 0.5 secs
  move 20 steps
  wait 0.5 secs
  move 20 steps
  wait 0.5 secs
  move 20 steps
  wait 0.5 secs
  move 20 steps
  wait 0.5 secs
  move 20 steps
  wait 0.5 secs
  turn ↻ 90 degrees
  play drum 1 for 0.25 beats
  play note 60 for 0.5 beats
```

Fig. 7.13 Scratch Program

3. Playing Sound in Scratch:

We can add sounds in our projects. It is very simple and easy. We can play different sounds like beat the drum or play different notes. You can use these sounds to make any animation, game or even just creating music. You can find complete list of notes and beats on following link. https://en.scratch-wiki.info/

- In this sample program, notes are used to create the music of a song.
- Try to develop this program and write the output of this program.

OUTPUT

```
when space key pressed
say I can play music
repeat 2
    play note 55 for 0.5 beats
    play note 55 for 0.5 beats
    play note 57 for 0.5 beats
    play note 55 for 0.5 beats
    play note 72 for 0.5 beats
    play note 71 for 0.5 beats
say Happy Birthday!
repeat 1
    play drum 1 for 0.25 beats
    play drum 2 for 0.25 beats
    play drum 3 for 0.25 beats
    play drum 4 for 0.25 beats
```

Fig. 7.14 Playing Sound

4. Changing the Backdrops:

By default, the Backdrop of stage is white/ blank. It can be changed. Place the cursor on "Choose a Backdrop" button and click. It consists of four different options:

- **Choose a backdrop from library:**
  You can choose a backdrop from available library of backdrops.
- **Paint New backdrop:**
  You can also paint a new backdrop
- **Upload backdrop from file:**
  An image on your computer can also be used as backdrop

- **New backdrop from Camera:**

  You can also capture a picture through webcam and use it as backdrop.

After adding a backdrop, you can use it in your program. Let's practice this skill. For developing this program, you need to add two backdrops in your program; add Blue Sky and Desert backdrops. Also delete cat from Sprite List and add Dinosaur1 as Sprite.

By using relevant blocks, try to develop this illustrated program.

After finishing coding, run this program. If you have done all things right, you will find a dinosaur, moving lazily on your stage and as it crosses the stage, the other Backdrop (Desert) appears on the stage and the dinosaur feels happy.

Now, try to understand each line of your program and in your notebook write the purpose of each command.

```
when [flag] clicked
set x to -127
set y to 0
switch backdrop to blue sky
repeat 10
    switch costume to bat1-b
    wait 0.2 secs
    move 30 steps
    switch costume to bat1-a
    wait 0.2 secs
switch backdrop to desert
set x to -127
set y to 0
repeat 20
    switch costume to bat1-b
    wait 0.2 secs
    move 15 steps
    switch costume to bat1-a
    wait 0.2 secs
```

Fig. 7.15 Changing Backdrop

5. Using Multiple Sprites:

We can have multiple sprites in our project. At least one sprite is always needed for the project. Just like New Backdrop, there are four ways to add different Sprite in our project. These are; **Choose sprite from library, Paint New Sprite, Upload Sprite from file and New Sprite from Camera.**

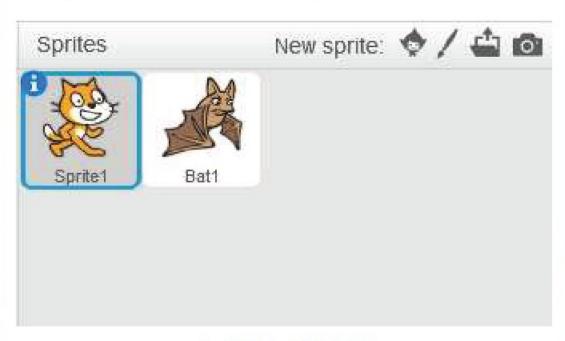



Fig. 7.16 Using Multiple Sprites

Here is a program in which two Sprites are used. Follow these steps to develop this program.
Add Bat as the second Sprite in you Sprite List.
This time you need to write codes for both Sprites.
Write the following code for each Sprite.

Bat1

For Bat 1

Sprite1

For Sprite 1

Fig. 7.17 Multiple Sprites Program

To run this program, click on flag icon. Then press the Space Key from keyboard. Observe the output of this program and write it in your notebook. In a group, discuss the logic of this program with your class-fellows. Remember, using multiple Backdrops and/or Sprites make your program more interactive and interesting.

6. Taking Input:

Scratch also allows developer to take input from the user. This illustration shows a simple program to take the input. In this program you can use *ask* command in which you may write your message, like in this example, name of user is asked. The third line is important and you need to add *say* command and join the *answer* command with this. The input that you take in *ask* is stored in *answer* and it is displayed using *say*.

Fig. 7.18 Taking Input

Let's develop a program.

1. First line will start the program.

2. you can find *ask* and *wait* commands in **Sensing Block**. You may add your message in these code blocks.

Fig. 7.19 Program for Input

3. You can find *set-to* command in **Data block.** For this program you will also need to create two variables from **Data Block**. For creating a variable click on Make a Variable button. A message box will appear which will ask you for a name of the variable. Enter a name to create the variable. Drag *set* command and set variable names. Also add answer from **Sensing Block.** This line will store the value given by the user into variable called "Obtained Marks".

Fig. 7.20 Creating Variable

4. Same as Line 2.

5. Same as Line 3.

6. You can add say command from **Looks Block** and enter your message.

7. For creating this code, first add multiplication operator ( ) * (100) from **Operator Block** and give the value 100. Now add Division Operators from same block ( ) / ( ) * (100). Now inside the both operands set the variable (Obtained Marks) / (Total Marks) * (100). Place this code inside say command.

8. Same as Line 6.

By understanding the logic of this program, now you can calculate many other things. Remember, developing program is great activity and coding is going to be an important skill in future. Through some basic skills that you learned in this unit, you can develop many complex projects by using your creativity and imagination.

- Scratch is a programming language. It includes animations and games.
- We can create interactive stories, games, animation, music, art, and presentations.
- There are two basic components of a Scratch program; Sprite and Script.
- Scratch Environment includes: Stage or Stage Preview Window, Script Area, Sprite List, Backdrop, and Script Block.
- We can use Scratch online and offline.
- The Stage is where you see your stories, games, and animations come to life.
- The Script is the set of stepwise instructions that users give to the sprite to do a particular task.
- Backdrop is the background that the user can add on stage.
- Scratch blocks are organized into different categories in the left columns.
  - Motion block has instructions to make the Sprite move, such as number of steps to take, direction of motion, etc.,
  - Event is used to trigger code at a specific time or action.
  - Sound is used to play different sounds.
  - Looks is used to change the appearance of the Sprite and Backdrop.
  - Control is used to control the action on the stage.
  - Sensing is used to sense any specific happening.
- Backdrop is the background of stage.
- Costume is the appearance of Sprite.
- Scratch has a 2D coordinate system: "x position" and "y position".
- User can choose Backdrop or Sprite from library, Paint New, Upload from File and Capture from Camera.
- Variables can be created through Data Pallet.

A. ENCIRCLE THE CORRECT ANSWER:

i) The feature of Scratch is ______________________.

a. It is a visual | b. Its free forever
c. No need to remember codes | d. All of above

ii) In Scratch, the character which moves on the Stage is called a ______.

a. Sprite | b. Command
c. Script | d. Event

iii) Repeat 10, forever and if … then codes are available in ___________.

a. Motion | b. Control
c. Look | d. Sensing

iv) The Looks of Sprite can be changed by using ____________.

a. Backdrop Tab | b. Costume Tab
c. Script Tab | d. Control Tab

v) A Scratch Program is at least made up with ___________.

a. Many Sprites | b. One Sprite and Code
c. One Sprite | d. One Sprite and Backdrop

vi) To change the position of Sprite on screen, we use _____________.

a. Coordinates | b. Stage Information
c. Command Pallets | d. Costume Tab

vii) move 10 steps can be found under ____________.

a. Look | b. Motion
c. Sound | d. Control

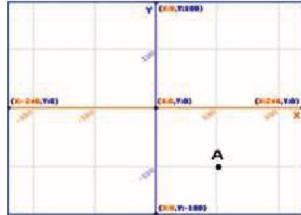
viii) Turn command, turns Sprite to specified:

Coordinate | c) Steps
Degree | d) Seconds

ix) This repeat 10 command is available in ____________________.

a. Event | c. Control
c. Motion | d. Looks

x) In the given picture identify the x and y coordinates (positions) of Point – A.

a. x=100, y=-100
b. x=-100, y=100
c. x=100, y=100
d. x=-100, y=-100

B. RESPOND THE FOLLOWING:

1. Explain the following:
   i. Script
   ii. Sprite
   iii. Backdrop
2. State the difference between repeat 10 and forever commands.
3. Write the use of the following codes: forever, wait, say, play sound, go to x, y
4. What is the difference between using Scratch online and offline?
5. In your notebook mark the color of each pallet in Script Tab. One is done for you. How are these colors helpful for the user?

	Pen

LAB ACTIVITIES

1. Divide the class into groups. Make two sprite using photos of two students and write scripts for them to do an activity of your choice. Use your imagination to make them do actions by using motion and control blocks.
2. Draw a matrix of 480 × 360 and then mark these points on the that matrix:
   a. x= -170, y =100
   b. x=110, y=190
   c. x= -120, y= -120
   d. x= 150, y= 0

   Now with the help of *go to-x-y* command, check your marks on the matrix.
3. Make a list of your 5 favorite Backdrops and 5 favorite Sprites.
4. Make a sample program in which two Sprites talk with each other. Try making few jokes.
5. Develop a program to enter Radius of Circle; calculate area and circumference of the circle and display the result.
6. In a group, try to develop a story by using multiple Sprites and Backdrops.
7. If available, explore few ideas from https://scratch.mit.edu/ideas and make programs accordingly. You can also download Coding Cards and Starter Projects from this website.

Teachers Note	Teachers should encourage students to explore different resources and learn further themselves.